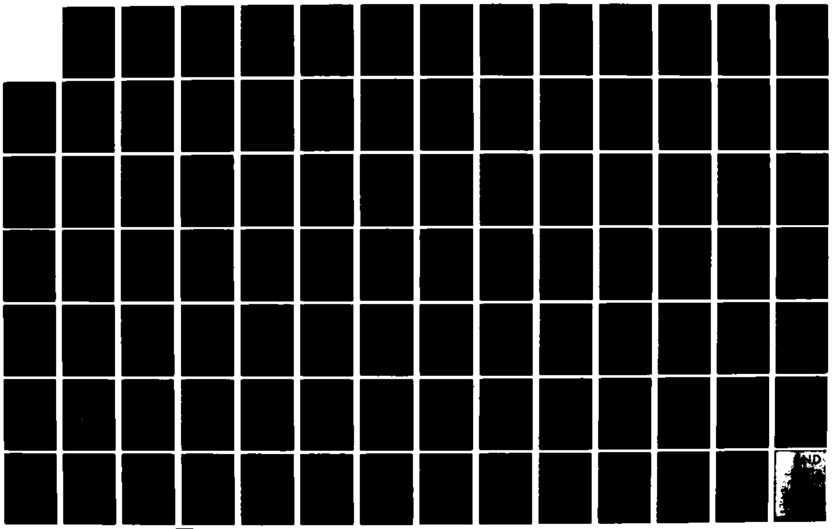
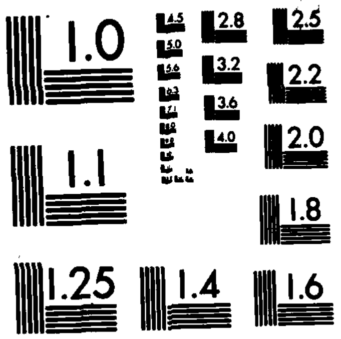
AD-A126 874 AN ADAPTIVE FINITE DIFFERENCE METHOD FOR HYPERBOLIC SYSTEMS ON ONE SPACE DIMENSION REVISION(U) CALIFORNIA UNIV BERKELEY LAWRENCE BERKELEY LAB J H BOLSTAD DEC 82 1/1

UNCLASSIFIED LBL-13287-REV F/G 12/1 NL




1.0
4.5
5.0
5.6
6.3
7.1
8.0
2.8
3.2
3.6
4.0
2.5
2.2
2.0
1.1
1.8
1.25
1.4
1.6

LBL-13287-rev. (15)

AN ADAPTIVE FINITE DIFFERENCE METHOD FOR HYPERBOLIC SYSTEMS IN ONE SPACE DIMENSION[1]

Abbreviated Title:
AN ADAPTIVE METHOD FOR HYPERBOLIC SYSTEMS

John H. Bolstad

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

present address:
Applied Math 217-50
California Institute of Technology
Pasadena, California 91125

December, 1982

Submitted to

SIAM Journal on Scientific and Statistical Computing

[1] Supported in part by the Office of Naval Research under contract N00014-75-C-1132, and by the Director, Office of Basic Energy Sciences, Engineering, Mathematical, and Geosciences Division of the U. S. Department of Energy under Contract DE-AC03-76SF00098.

AN ADAPTIVE FINITE DIFFERENCE METHOD FOR HYPERBOLIC SYSTEMS IN ONE SPACE DIMENSION

Abstract. Many problems of physical interest have solutions which are generally quite smooth in a large portion of the region of interest, but have local phenomena such as shocks, discontinuities or large gradients which require much more accurate approximations or finer grids for reasonable accuracy. Examples are atmospheric fronts, ocean currents, and geological discontinuities.

In this paper we develop and partially analyze an adaptive finite difference mesh refinement algorithm for the initial boundary value problem for hyperbolic systems in one space dimension. The method uses clusters of uniform grids which can "move" along with pulses or steep gradients appearing in the calculation, and which are superimposed over a uniform coarse grid. Such refinements are created, destroyed, merged, separated, recursively nested or moved based on estimates of the local truncation error. We use a four-way linked tree and sequentially allocated deques (double-ended queues) to perform these operations efficiently. The local truncation error is estimated using a three-step Richardson extrapolation procedure in the interior of the region, and differences at the boundaries. Our algorithm was implemented using a portable, extensible Fortran preprocessor, to which we added records and pointers.

The method is applied to two model problems: the second order wave equation with counterstreaming Gaussian pulses, and the Riemann shock-tube problem. For both problems our algorithm is shown to be three to five times more efficient (in computing time) than the use of a uniform coarse mesh, for the same accuracy. Furthermore, to our knowledge, our algorithm is the only one which adaptively treats time-dependent boundary conditions for hyperbolic systems.

Key words. hyperbolic systems, adaptive, mesh refinement, finite differences, local truncation error, partial differential equations, Richardson extrapolation, wave equation, Riemann shock tube, data structures, macro preprocessor.

1. Introduction. Many problems of physical interest have solutions which are smooth in a large portion of the region of interest, but have local phenomena such as shocks, discontinuities or large gradients which require much more accurate approximation or finer meshes for reasonable accuracy. Examples of this are atmospheric fronts, ocean currents, geological discontinuities, and storm surges.

When the positions of the gradients are known *a priori*, and are independent of time, one can use coordinate transformations, a technique used extensively in aerodynamic computations, *e.g.*, Steger and Chaussee [44]. If the position of the gradients changes as an *a priori* function of time, the mapping function can change with time (*e.g.*, flow past a helicopter blade). However, when the manner in which the gradients move is not known in advance, this technique cannot be used.

The use of a fine mesh throughout the entire calculation region usually requires too much computer time and/or storage. An alternative method is to use an underlying coarse mesh for the entire region, and to superimpose a fine grid, or grids, on the region(s) where the solution is varying rapidly. The crucial difficulty is that the refined region(s) must then move along with the rapidly varying portion of the solution, at all times enclosing this portion.

The necessity for this was illustrated by Browning, Kreiss and Oliger [7]. They computed the numerical solution of an initial boundary value problem whose exact solution was a rapidly oscillating sine wave. The wave was accurately represented on the fine part of the spatial grid, but when it passed into the coarse mesh, it was almost completely obliterated. From this it is clear that the rapidly varying part of the solution must not be allowed to escape the refinement region.

Adaptive methods have been used in other areas of numerical analysis for many years; some references are given in [5]. The study of adaptive algorithms for time-dependent partial differential equations is a very active research area. We will now list several properties which distinguish our method from others, and at the same time list a few of the many references to other work.

We should first mention that the methods of Gropp [18], Berger [2] and the author all follow the strategy of Budnik and Oliger [8], Oliger [33], and Berger, Gropp and Oliger [3]. Gropp's paper is a test of the feasibility of this method in two dimensions. Berger's method is for hyperbolic problems in one and two space dimensions. It uses the same method as ours for placing refinements (a method which is more general than Gropp's) and, like ours, allows fine grids to be recursively nested. Berger's method allows refinements to have arbitrary orientation in two dimensions. Her method does not treat boundary conditions as ours does. Our data structure is somewhat different than hers, but cannot be generalized to more than one space dimension.

The most important difference between our algorithm and most others is in efficiency. On model problems our algorithm is three to five times more efficient than using a uniform mesh which produces the same accuracy. (Berger obtains similar efficiency factors, while Gropp's were somewhat less.) Most other methods fail to discuss this matter, and fail to give computer times for their methods. An exception is Hu and Schiesser [24], who found their method to be approximately 1.5 times as efficient on a test problem.

The second distinguishing feature of our method is that it is designed for hyperbolic systems. Many other methods claim to work on both parabolic and hyperbolic problems, but their examples are only given for the former. In contrast, our method can be used for parabolic problems, but it is not yet efficient for them since it uses explicit time steps. Our method does not require us to know the position or the direction of characteristics.

We can classify adaptive algorithms as either (a) moving (or deformed) grid methods, or (b) local refinement methods. In the former, a fixed number of grid points are present, and they move with the solution (possibly via coordinate transformations). In more than one space dimension this can lead to distorted grids. The other approach uses a stationary coarse grid, on which are overlayed or interpolated finer grid points. To follow a wave, grid points are created or destroyed as necessary, but grid points are not actually moved. Our method is a local refinement method, and our refined grids can be created, destroyed, merged, separated or recursively nested. In some problems there is a shock or gradient that needs to be resolved at one time, and several at another time (for example, see our Riemann shock tube problem in Section 8). A method with a fixed number of points may use too many or too few part of the time. If too few are used, the grid points may also undergo sudden transitions when another steep gradient appears. Hedstrom and Rodrigue [23] discuss other advantages and disadvantages of these approaches. The methods of Davis and Flaherty [11], Miller [17], [32] Dwyer, Kee and Sanders [13], Rai and Anderson [39], Brackbill and Saltzman [6], and White [47] are moving grid methods. The methods of Berger, Gropp, Gannon [16], Hu and Schiesser Lam and Simpson [30], and the author are local refinement methods.

Among the local refinement methods we can distinguish those in which fine grid points are interpolated, or patched, into the coarse grid, from those in which the fine grid points overlay (and are separate from) the coarse grid. The methods of Lam and Simpson and Hu and Schiesser are examples of the former, while the methods of Gropp, Berger and the author all use the latter approach. An advantage of the overlay approach is its adaptability to parallel or pipeline processors. The solution can be advanced (and the error estimated) almost independently on different grids.

The fourth distinguishing feature of our method is that it refines the mesh in both space and time. Most other adaptive methods use a time step which is the same at every spatial mesh point at a given time. Since we use explicit methods, it is essential to allow finer time steps in refinements than in the coarse mesh. Otherwise, stability would force us to use tiny time steps over the whole spatial region. To our knowledge, only the methods of the author, Berger, and Gropp allow this.

Closely related to this is the fifth distinguishing feature of our method: the use of nontraditional (for numerical methods) data structures. This is necessary to efficiently implement variable numbers of refined grids. Berger has implemented more complicated data structures in two space dimensions. Other adaptive methods using nontrivial data structures were developed by Gannon and by Rheinboldt and Mesztenyi [40].

A sixth notable feature of our method is our criterion for mesh placement. We arrange our mesh points to "approximately equidistribute" the local truncation error (de Boor [12] and Pereyra and Sewell [37]). Other methods use measures such as the gradient, arclength, or second derivative. We feel that these *ad hoc* methods may work in certain instances, but are not sufficiently general. Other methods that use the local truncation error are those of Pierson and Kutler [38], Rai and Anderson, and Berger. Gannon uses an analogous criterion [1] for finite elements.

A unique feature of our method is the adaptive treatment of time-dependent boundary conditions. Implementing this requires local error estimates at the boundary, a variable number of grid points, and nontrivial data structures. This is obviously important in applications like limited area weather forecasting.

The final distinguishing feature of our algorithm is its method of implementation. Fortran is inconvenient because it lacks data structures and control structures, and other languages have portability problems. So we used an extensible, portable Fortran preprocessor, to which we added pointers and records. Stein [45] and Gropp [19] have also considered languages for adaptive algorithms.

We now summarize what is contained in the rest of this paper. In Section 2 we describe our adaptive mesh-refinement algorithm in detail. We first describe the continuous problem, the usual first order hyperbolic system on a strip in one space dimension. Next we describe our mesh structure. A detailed description of the algorithm is then provided, including techniques at boundaries and at interfaces between coarse and fine meshes. Section 3 discusses the estimation of the local truncation error in the interior of refinements, at coarse/fine interfaces, and at boundaries.

In Section 4 we give a brief discussion of stability. Section 5 summarizes partial results in [5], which show how mesh refinement affects the rate of convergence. Section 6 describes the data structures we used to implement the algorithm. Section 7 discusses the Fortran preprocessor language we used. Section 8 provides computational results. As model problems we used the first order wave equation, the second order wave equation with counterstreaming Gaussian pulses, and the Riemann shock tube problem.

2. Mesh Structure and Solution Algorithm.

In this section we describe the differential equations to be approximated. Then we give a recursive description of our mesh structure and of our algorithm for advancing the solution.

2.1. The Continuous Problem.

Let Ω denote the spatial interval $a \le x \le b$. For purposes of analysis we will consider a linear first order, one (space)-

dimensional, $n \times n$ hyperbolic system

$$Lu \equiv u_t - A(x,t)u_x - B(x,t)u = F(x,t), \tag{2.1}$$

on a "vertical" strip $\Omega \times \{t \geq 0\}$, with initial condition

$$u(x,0) = f(x), \qquad x \in \Omega, \tag{2.2}$$

and boundary conditions

$$u^{\mathrm{I}}(a,t) = S_{\mathrm{II}}(t)u^{\mathrm{II}}(a,t) + g_1(t), \qquad t \geq 0, \tag{2.3}$$

$$u^{\mathrm{II}}(b,t) = S_{\mathrm{I}}(t)u^{\mathrm{I}}(b,t) + g_2(t), \qquad t \geq 0. \tag{2.4}$$

Here A and B are $n \times n$ matrices and F is an n-vector. We have, as usual, assumed that A has already been transformed into diagonal form by a nonsingular uniformly bounded similarity transformation $T(x,t)$, so that

$$A = \begin{pmatrix} A^{\mathrm{I}} & 0 \\ 0 & A^{\mathrm{II}} \end{pmatrix}$$

with $T(x,t)$ and $T(x,t)^{-1}$ uniformly bounded, and

$$A^{\mathrm{I}} = \mathrm{diag}(\kappa_1, \kappa_2, \ldots, \kappa_J) < 0,$$
$$A^{\mathrm{II}} = \mathrm{diag}(\kappa_{J+1}, \kappa_{J+2}, \ldots, \kappa_n) > 0,$$
$$u^{\mathrm{I}} = (u_1(x,t), u_2(x,t), \ldots, u_J(x,t))^T,$$
$$u^{\mathrm{II}} = (u_{J+1}(x,t), u_{J+2}(x,t), \ldots, u_n(x,t))^T,$$

and

$$S_{\mathrm{I}} \in C^{(n-J)\times J}, \; S_{\mathrm{II}} \in C^{J\times(n-J)}.$$

By far the most important restriction is that our problem has only one space dimension. Even in two space dimensions the problem has severe additional difficulties, such as irregular geometries, orientation of refinements, pattern recognition, the need for more complicated data structures, and topology of boundaries. The restriction to hyperbolic behavior insures that we can use explicit time steps. This assumption greatly simplifies both the error estimation and the manipulation of moving meshes. However, many computational problems in fluid dynamics and elsewhere are of this type.

We have assumed that the matrix A is in diagonal form because this makes it easier to write down boundary conditions (2.3)-(2.4) which yield a well-posed problem. We have assumed the problem is linear so that we can analyze local error estimation and convergence (in fact, we shall assume constant coefficients in Sections 3 and 5). Neither of these assumptions is necessary in practice, as shown by computations in Section 8.

We assume that the overall phenomena being studied are such that, except for relatively small regions, a coarse uniform mesh is sufficient to resolve them. We further assume these small regions change with time in a way which cannot conveniently be determined *a priori*. We also assume that, at any time, these small regions are approximately the same for all solution components. In other words, if the differential equations describe velocity and pressure, then large pressure gradients occur in approximately the same regions where large velocity gradients occur. Thus we can use the same refinements for each component of the solution vector. (The assumptions in this paragraph are necessary only for efficiency. The method will work without them, but it might refine too large a portion of the region.)

For purpose of analysis we assume that the solution is smooth. This means that there are no corner discontinuities, *i.e.*, $u^{I}(a,0) = f^{I}(a)$ and $u^{II}(b,0) = f^{II}(b)$. Furthermore, it means there are no shocks present. These assumptions enable us to estimate the local truncation error using asymptotic expansions. These restrictions are not always necessary in practice (see problem P3 in Section 8), but then our method is not supported by analysis.

2.2. Mesh Structure. We will now formally describe our refined grids. We will compute on a basic rectangle $R = \Omega \times [\quad T]$ wh[illegible] is subdivided into subrectangles on each of which is imposed a uniform grid (see Fig. 1).

We will proceed recursively by "levels of refinement". The word *level* in this context refers not to the time level, but to how fine a grid spacing is. Finer grids will have higher levels. Let Λ be the maximum level. On each level $l = 0, 1, \ldots, \Lambda-1$ we will introduce a finite number of space-time *refinement rectangles* or boxes B_ν^l contained in rectangle R. (All such rectangles will be *solid*, that is, they include both interior and boundary.) Each such rectangle will have sides parallel to the coordinate axes, and for $l \geq 1$ each l-th level rectangle must lie entirely in an $l-1$-st level rectangle. Furthermore, no two l-th level rectangles can overlap. The boundary of each l-th level rectangle will be the boundary of a uniform $l+1$-st level (space-time) grid. All $l+1$-st level grids will have the same space and time steps. Loosely speaking, an $l+1$-st level refinement is one of these grids viewed at a fixed time.

To prime the recursive pump, we will define the *zero-th level spatial division points* of the interval $[a, b]$ as the sequence of points $<x_0^0 = a, x_1^0 = b>$. Similarly, the *zero-th level time division points* of the interval $[0, T]$ comprise the sequence $<t_0^0 = 0, t_1^0 = T>$. Let $h_0 = b - a$ and $k_0 = T$ be the *zero-th level space* and *time steps*, respectively. We define U^0 as the set of four corner points of the rectangle R.

For $l = 0, 1, \ldots, \Lambda-1$ we now form the *l-th level partition* P_l of $[0, T]$, which is a subsequence of the time division points $<t_m^l>$:

$$0 = t_0^l < t_{m_1}^l < t_{m_2}^l < \cdots < t_{m_{s_l-1}}^l < t_{m_{s_l}}^l = T. \tag{2.5}$$

We have omitted from the notation the dependence of the subsequence $<m_i>$ on l. For $l = 0$, P_0 is identical to the sequence $<t_m^0>$ of time division points. Thus $s_0 = 1$ and $m_1(0) = 1$. For $l \geq 1$, P_l must contain as a subsequence the points in the partition P_{l-1}.

This partition divides the region R into *l-th level* horizontal *strips* S_i^l, $i = 1, 2, \ldots, s_l$. For $l = 0$ the only such strip S_1^0 is identical to the rectangle R. For $l \geq 1$ each of these strips is contained in an $l-1$-st level strip, since P_{l-1} is a subsequence of P_l. The partition points are the times when we adjust the mesh. They must be chosen before the solution of the problem. (The partitions and strips for $l > 1$ could be dispensed with if we never adjust the mesh *between* coarse time steps.) For $l = 1$ we have shown in Fig. 1 the time division points $<t_m^1>$ and the partition P_1 with $m_0^1 = 0$, $m_1^1 = 2$, $m_2^1 = 3$, and $m_3^1 = 5$. We shall denote the partition points P_1 alternatively as

$$0 = t^0 < t^1 < t^2 < \cdots < t^s = T, \tag{2.6}$$

and denote the first level strips S_i^1 by S_i. Thus $S_i = \Omega \times [t^{i-1}, t^i]$ for $i = 1, \cdots, s$.

We will now introduce a set of zero or more nonoverlapping *l-th level* (solid) *refinement rectangles*

$$\{B_\nu^l, \nu = 1, 2, \ldots, g_l\}.$$

(If $g_l = 0$ the recursion ends.) There is only one zero-th level rectangle B_1^0, and it is identical to the rectangle R. For $l \geq 1$, each rectangle B_ν^l is required to lie entirely in some $l-1$-st level refinement rectangle B_π^{l-1}. The latter will be called a *parent* of the former. Each such rectangle B_ν^l will have horizontal sides whose t-coordinates are required to be *adjacent* members of the partition P_l (2.5). That is, the horizontal sides of the rectangle are the same as the the horizontal sides of the l-th level strip in which it is contained. Since P_{l-1} is a subsequence of P_l for $l \geq 1$, we are guaranteed that B_ν^l is "vertically contained" in its parent.

For $l \geq 1$, the x-coordinates of the vertical sides of rectangle B_ν^l can be any l-th level spatial division point, so long as B_ν^l is "horizontally contained" in its parent. In other words, let its parent B_π^{l-1} have left and right vertical sides with coordinates

$$x = x^{l-1}_{\alpha(\pi)} = x^{l}_{N^{(l-1)}\alpha(\pi)} = a + h_{l-1}\alpha(\pi)$$

and

$$x = x^{l-1}_{\omega(\pi)} = x^{l}_{N^{(l-1)}\omega(\pi)} = a + h_{l-1}\omega(\pi),$$

respectively. (Here $\alpha(\pi)$ and $\omega(\pi)$ are nonnegative integers.) Then for the coordinates $x^{l}_{\alpha(\nu)}$ and $x^{l}_{\omega(\nu)}$ of the left and right vertical sides of rectangle B^{l}_{ν}, we require

$$x^{l-1}_{\alpha(\pi)} < x^{l}_{\alpha(\nu)} < x^{l}_{\omega(\nu)} < x^{l-1}_{\omega(\pi)}, \tag{2.7a}$$

i.e.,

$$N^{(l-1)}\alpha(\pi) < \alpha(\nu) < \omega(\nu) < N^{(l-1)}\omega(\pi). \tag{2.7b}$$

If the left (right) edge of the parent rectangle lies on the left (right) edge of the region R, tnen we allow the leftmost (rightmost) inequality to become "<=".

For $l \geq 0$, let $N^{(l)}$ and $M^{(l)}$ (the l-th level spatial and time *refinement ratios*) be integers greater than one. (Fig. 1 shows the case $N^{(l)} = 3$ and $M^{(l)} = 2$ for $l \geq 1$.) Let $h_{l+1} = h_l / N^{(l)}$ and $k_{l+1} = k_l / M^{(l)}$ be the *l+1-st level space* and *time steps*, respectively. We now define the sequences of (uniform) *l+1-st level spatial* and *time division points*

$$<x^{l+1}_j = a + jh_{l+1}: j = 0,1, \ldots, \prod_{\mu=0}^{l} N^{(\mu)}>,$$

and

$$T^{l+1} = <t^{l+1}_m = mk_{l+1}: m = 0,1, \ldots, \prod_{\mu=0}^{l} M^{(\mu)}>,$$

of the intervals Ω and $[0, T]$, respectively. They are respectively $N^{(l)}$ and $M^{(l)}$ times as fine as the l-th level ones. The set of all points

$$U^{l+1} = \{(x^{l+1}_j, t^{l+1}_m)\}$$

occupies the entire rectangle $R = \Omega \times [0, T]$. The subset of these points contained in the (solid) refinement rectangle B_ν^l is defined to be the $(l+1)$-st *level* (space-time) *grid* G_ν^{l+1} occupying B_ν^l. More specifically, if B_ν^l occupies the l-th level horizontal strip S_i^l, then G_ν^{l+1} consists of that subset of U^{l+1} whose x components have subscripts

$$j = \alpha(\nu)N^{(l)},\ \alpha(\nu)N^{(l)}+1,\ \ldots,\ \omega(\nu)N^{(l)}, \tag{2.8}$$

and whose t components have subscripts

$$m = m_{i-1}(l)M^{(l)},\ m_{i-1}(l)M^{(l)}+1,\ \ldots,\ m_i(l)M^{(l)}.$$

(Recall that the subsequence $<m_i>$ depended on the level l.) This completes our recursive definition.

Now we come to the most important definition of this paper.

DEFINITION 1. Let G_ν^{l+1}, $l = 0, 1, \ldots, \Lambda-1$, be an $l+1$-st level grid, occupying an l-th level rectangle B_ν^l, whose mesh points are as given above. Let t be any time such that

$$t^l_{m_{i-1}(l)} \le t \le t^l_{m_i(l)}, \tag{2.9}$$

and let t_m^{l+1} be the greatest $l+1$-st level time division point not exceeding t. An *l+1-st level refinement* at time t, corresponding to B_ν^l or G_ν^{l+1}, is a sequence of ordered pairs

$$R_\nu^{l+1}(t) = <(x_j^{l+1},\ v_j^{l+1}(t_m^{l+1}))>,$$

where j has the values (2.8). The first components comprise the sequence of $l+1$-st level spatial division points contained in the horizontal sides of the refinement rectangle B_ν^l (equivalently, the sequence of x components of the grid points in G_ν^{l+1}); the second components are the approximate solution values evaluated at these spatial points, but at time t_m^{l+1}. Here $v_j^{l+1}(t)$ is an approximation to the vector $u(x_j^{l+1}, t)$.

An important property of our definition is that an $l+1$-st level refinement exists not only at $l+1$-st level time division points T^{l+1}, but also at "finer" time division points $T^{l+2}, \ldots, T^{\Lambda}$ satisfying (2.9). However, solution values for an $l+1$-st level refinement are only updated at $l+1$-st level time division points.

For $l \geq 1$ let B_{ν}^{l} be any refinement rectangle, and R_{ν}^{l+1} its corresponding refinement. A vertical side of B_{ν}^{l} which does not lie on the boundary of the region R will be called a *coarse/fine interface*. Similarly, the left or right endpoint of R_{ν}^{l+1} will also be called a coarse/fine interface if it does not lie on the left or right boundary of the region R.

The first level (or *coarse*) space-time grid occupies the whole rectangle $B_{1}^{0} = R$. Hence, the first level, or coarse, refinement is present at all times, and higher level refinements are considered to be superimposed on it. (Strictly speaking, we should not call this a refinement, since it doesn't refine anything. We use this terminology to avoid special cases.) We will assume as given the largest wave propagation speed. This is usually known by the problem originator, and determines the spacing of the coarse refinement.

Another factor which must determine the spacing of the coarsest refinement is the wavelength of any "background disturbances" to the phenomenon of interest (see our model problem P1 later in this chapter for an example). This too is assumed known; for guides to the number of mesh points needed per wave length, see Kreiss and Oliger [29].

We will now discuss some further restrictions imposed on our refinement rectangles. We will require that no two l-th level refinement rectangles in the same l-th level horizontal strip can intersect or abut. (But l-th level rectangles in adjacent strips may abut.) Assume an l-th level strip contains two l-th level rectangles B_{μ}^{l} and B_{ν}^{l}, having left and right vertical sides with x coordinates

$$x^l_{\alpha(\mu)},\ x^l_{\omega(\mu)} \text{ and } x^l_{\alpha(\nu)},\ x^l_{\omega(\nu)},$$

respectively. Without loss of generality, assume that the left side of the former is to the left of the left side of the latter, $\alpha(\mu) < \alpha(\nu)$. Then

$$\omega(\mu) < \alpha(\nu).$$

This is no restriction in practice; if two such rectangles overlap or abut, we simply consider them to be one rectangle.

Let $\lambda_l = k_l / h_l$. Then our construction ensures that λ_l is a constant depending on l. For simplicity, our implementation restricts the refinement ratios for $l \geq 1$ to be the same, *i.e.*, $N^{(l)} = N$ and $M^{(l)} = M$, for $l = 1, 2, \ldots, \Lambda-1$. This condition is not essential, but it poses no real restriction, as we will see in Section 8.3. For convergence studies, we shall in addition assume that $M = N$, so that λ_l = constant, independent of l.

In Sections 4 and 5 we assume that we adjust the mesh only at coarse time steps. Thus all partitions P_l are the same as the first level partition. In practice, we can adjust the mesh more often (or less often) than every coarse time step, but for each l we always adjust at times which are a multiple of the l-th level time step. That is, every partition P_l is of the form

$$0 = t^l_0 < t^l_\vartheta < t^l_{2\vartheta} < \cdots < t^l_{(s_l-1)\vartheta} < t^l_{s_l\vartheta} = T,$$

where $\vartheta = \vartheta(l)$ is a small positive integer.

The blackened rectangle in Fig. 1 illustrates the possibilities. If we adjust the mesh only at coarse time steps, then this rectangle contains no mesh points in its interior. However, if we allow mesh adjustment between coarse time steps, then the blackened rectangle may contain six subrectangles.

2.3. Operations on Refinements. We now describe the operations we can perform on refinements. For simplicity, we shall assume that we adjust the mesh only at coarse time steps, and use the notation (2.6) for the division points between strips.

We shall say that two refinements are ***equivalent*** when their first components (x coordinates) are the same, regardless of the time or the solution values. Thus, for all times (2.9) encompassed by the refinement rectangle B_ν^l, the refinements $R_\nu^{l+1}(t)$ corresponding to B_ν^l are equivalent. In this sense, we can say that to each rectangle B_ν^l or grid G_ν^{l+1} there corresponds ***one*** refinement. This equivalence concept is useful for describing refinement manipulations which do not depend on the differential equation calculations. Clearly, only refinements with the same level can be equivalent.

Suppose first that there is an l-th level refinement rectangle B_μ^l $(l > 0)$ contained in the strip S_i (2.6). Assume that the horizontal sides of B_μ^l occupy the interval

$$x_{\alpha(\mu)}^l \le x \le x_{\omega(\mu)}^l.$$

Also assume that no part of any l-th level refinement rectangle in strip S_{i-1} lies in this interval. Then we will say that the refinement R_μ^{l+1} corresponding to B_μ^l has been ***created*** at time $t = t^{i-1}$. Similarly, if we replace S_{i-1} by S_{i+1} and t^{i-1} by t^i, we say that R_μ^{l+1} has been ***deleted*** at time t^i.

Now suppose there are two l-th level refinement rectangles $B_\mu^l \subset S_i$ and $B_\nu^l \subset S_{i+1}$. According to our definition, the refinement R_μ^{l+1} corresponding to B_μ^l only exists for $t^{i-1} \le t \le t^i$, and the refinement R_ν^{l+1} corresponding to B_ν^l only exists for $t^i \le t \le t^{i+1}$. We will now examine the possible relationships between these refinements.

Suppose the rectangles B_μ^l and B_ν^l have the same left and right sides, $\alpha(\mu) = \alpha(\nu)$ and $\omega(\mu) = \omega(\nu)$. Then the first components of the refinements corresponding to these rectangles are the same. By our definition, the refinements corresponding to B_μ^l and B_ν^l are equivalent. In this sense we may say that a single refinement now exists for times $t^{i-1} \le t \le t^{i+1}$.

Now suppose that the refinement rectangles are situated as before, but

$$\alpha(\mu) \le \alpha(\nu) < \omega(\nu) \le \omega(\mu),$$

(with at most one equality), and no part of any other l-th level refinement rectangle in strip S_{i+1} lies in the interval

$$x^l_{\alpha(\mu)} \le x \le x^l_{\omega(\mu)}.$$

Then we will say that the refinement R^{l+1}_μ has ***contracted*** at $t = t^i$ to form the refinement R^{l+1}_ν. By interchanging refinement rectangles and strips, respectively, an analogous definition can be given for an ***expanding*** refinement.

If B^l_μ and B^l_ν are situated as before, but

$$\alpha(\mu) < \alpha(\nu) < \omega(\mu) < \omega(\nu),$$

and no part of any other l-th level refinement rectangle in strips S_i or S_{i+1} occupies the interval

$$x^l_{\alpha(\mu)} \le x \le x^l_{\omega(\nu)},$$

then refinement R^{l+1}_μ has ***moved right*** at $t = t^i$ to become the refinement R^{l+1}_ν. Analogously, we can define what it means for a refinement to ***move left***.

Finally, suppose rectangle B^l_μ is in strip S_i as before, but strip S_{i+1} contains two (disjoint) l-th level refinement rectangles B^l_ν and B^l_π, with the former to the left of the latter. Assume that

$$\alpha(\nu) \le \alpha(\mu) < \omega(\nu) < \alpha(\pi) < \omega(\mu) \le \omega(\pi),$$

and that no part of any other l-th level refinement rectangle in strips S_i or S_{i+1} lies in the interval

$$x^l_{\alpha(\nu)} \le x \le x^l_{\omega(\pi)}.$$

Then the refinement R^{l+1}_μ is said to ***separate*** or ***split*** into refinements R^{l+1}_ν and

R_π^{l+1}. Analogous definitions can be given for two refinements to *merge* into a third.

The above are typical operations on refinements, but they do not exhaust the possibilities (for example, a refinement could split into three refinements, although this is quite rare). Fortunately, however, an exhaustive listing is not needed. All that is required is an algorithm which takes a set of l-th level refinements ($l > 1$) at time $t = t^i$, $i = 0, 1, \ldots, s-1$ and produces a new set of such refinements. For each l, once the left and right edges of the new refinements are determined (by local error estimates), this readjustment can be done in a single left-to-right scan of the existing l-th level refinements.

In Section 6 we will explain how these operations are implemented.

2.4. A Model Problem. We will now introduce our first model problem, which will be used in some of our computations in Section 8. It is the first order wave equation ("color equation")

$$\begin{aligned} &u_t = -cu_x, && a \le x \le b,\ 0 \le t,\ 0 < c, && \text{(P1)}\\ &u(x,0) = g(x), && a \le x \le b, &&\\ &u(0,t) = g(-ct), && 0 \le t, && \end{aligned}$$

with exact solution $u(x,t) = g(x-ct)$. We take $a = 0$, $b = 4$, and $c = 1$. The function g is taken to be a Gaussian pulse, traveling to the right with speed c, superimposed on a sinusoidal background,

$$g(x) = \exp(-\alpha(x+½)^2) + 0.1\sin 2\pi(x+½),$$

with $\alpha = 200$. The parameter α control the steepness and thickness of the pulse. For $\alpha = 200$, the pulse occupies about 8 percent of the interval $[0, 4]$. This models more realistic problems such as an atmospheric front or storm surge.

2.5. Solution Algorithm. We now describe our algorithm. As in the initial value problem for ordinary differential equations, we will need to give a tolerance δ on the local truncation error, which will be used to decide where to refine the mesh. (We have only used absolute error since all of our example problems vary between 0 and 1. In general, one should use a combination of relative and absolute error, as in Shampine and Gordon [42].)

For the initial value problem for o.d.e.'s, Stetter [46], and others, have shown how to estimate (but not control) the global error while the solution is being computed. This requires only a small amount of additional computation and storage for that case. Further investigation would be needed to apply this to the initial boundary value problem. But even if were done, one would still need to prescribe a local error tolerance.

Since our refinements are defined recursively, we can define the algorithm recursively. We will use an Algol-style notation. A non-recursive description of the same algorithm on a model problem was given in [5].

```
procedure advance.solution($l$, $t$);
  real $t$; integer $l$, $i$;
  value $t$;

  comment advance solution from $t$ to $t + M^{(l-1)}k_l$;
  for $i$ = 1 to $M^{(l-1)}$ do
    begin
    if mesh adjustment time at this level
        begin
        estimate.error($l$);
        if $l$ < max.level then adjust.mesh($l + 1$)
        end;
    if level $l + 1$ exists then advance.solution($l + 1$, $t$);
    compute the solution at time $t + k_l$ on all level $l$ refinements;
    Set $t \leftarrow t + k_l$
    end;
  if $l > 1$ copy (project) solution values from this
  refinement to its parent refinement at level $l-1$;
end advance.solution;

$l = 1$;
$t = 0$;
Obtain initial values on coarsest mesh (level 1);
```

advance.solution(l, t);

Thus, we advance on the highest level refinements first, then the next highest, etc. (One can also proceed from the coarsest level to the finest, and this may be advantageous in more space dimensions.) For example, if our problem has three refinements, (one each at levels 1, 2, and 3, respectively) and refinement ratios $N = M = 3$, we compute the solution at the next coarse time step by advancing the different levels in the following order: 3, 3, 3, 2, 3, 3, 3, 2, 3, 3, 3, 2, 1. (Notice that each recursive activation of our procedure has its own private copy of i and t, which may be different from the values of i and t at other levels. Thus t must be called by value and not by name or reference.)

We now explain the steps of the algorithm in more detail.

"Mesh adjustment time" means that the time level is one of those occurring in the partitions P_l given in Section 2.2. If the time t is a member of one of the partitions P_l, we estimate the local truncation error (see Section 3) that would be made *if* we took one forward time step in the level l refinement, but we do not actually take the step. Mesh points whose error estimate exceeds the local error tolerance are marked as needing refinement. These points are grouped into intervals. Several extra "buffer" mesh points are added to both ends of each such interval. This will be explained later.

In general, there may be more than one refinement at each level (except the first). In that case, the operations are done for all refinements on a given level, starting with the leftmost refinement.

To adjust the mesh, we compare the intervals just produced with the existing refinements. If these are not identical, refinements may have to be "moved", created, deleted, merged or separated. If a refinement on level l moves into a region formerly occupied only by a refinement on level $l-1$, we may need solution values that do not yet exist at mesh points in refinement l. These are obtained by linear or quadratic interpolation in space from solution values on the $l-1$-st level refinement.

Creation of a new refinement is done the same way, by spatial interpolation from its parent refinement. At any mesh adjustment time, an $l-1$-st level parent refinement can give birth to any number of l-th level refinements, but no higher level ones. An exception is made at $t = 0$. If refinement(s) of the coarse mesh are needed at that time, we obtain the new solution values directly from the initial function f rather than from interpolation. That is, we take as initial values

$$v_j^l(0) = f(a + jh_l), \tag{2.10}$$

where $j = 0, 1, \ldots, N^{(0)}$ when $l = 1$, and j has the values given in (2.8) for any other refinement introduced at $t = 0$. This allows us to add as many levels of refinement as are initially necessary. Thus, the method performs properly even when the initial mesh is "too coarse".

If a refinement occupies a spatial interval I, it can be deleted when it has no children, and the local error estimate of its parent in interval I is below the tolerance.

We now explain the "copy" operation. Certain points (x, t) of our region R (such as F in Fig. 2) are covered by mesh points in refinements at more than one level. (This is done for ease of implementation, and arises because we overlay, rather than patch or interpolate refinements.) We always want to use the solution values computed on the finest (highest level) mesh, so for these points we must project (copy) the values obtained on a level l refinement to the appropriate positions in the parent (level $l-1$) refinement.

It remains only to describe our difference approximations.

2.6. Difference Approximations. We first define the forward, backward, and centered difference operators D_+^l, D_-^l, and D_0^l, operating on l-th level refinements:

$$D_+^l v_\nu(t) = h_l^{-1}(E-1)v_\nu(t) = (v_{\nu+1}(t) - v_\nu(t))/h_l,$$
$$D_-^l v_\nu(t) = h_l^{-1}(1-E^{-1})v_\nu(t) = (v_\nu(t) - v_{\nu-1}(t))/h_l,$$
$$D_0^l v_\nu(t) = (2h_l)^{-1}(E-E^{-1})v_\nu(t) = (v_{\nu+1}(t) - v_{\nu-1}(t))/2h_l,$$

where we have omitted the superscript l on v.

To advance the solution one time step from t to $t+k_l$, we will treat separately the interior of a refinement, coarse/fine interfaces, and boundaries. In the interior of an l-th level refinement we will use explicit two (time)-level difference approximations to (2.1),

$$v_\nu^l(t+k_l) = Q_0 v_\nu^l(t) + k_l F_\nu^l(t), \tag{2.11}$$

where $t = t_m^l$,

$$Q_0 = Q_0(l) = \sum_{j=-r}^{q} A_j(x_\nu^l + jh_l,\, t,\, h_l)E^j\ ,$$

$E = E(l)$ is the shift operator

$$E_j v_\nu^l(t) = v_{\nu+j}^l(t),$$

q and r are nonnegative integers, $v_\nu^l(t)$ is an approximation to $u(x_\nu^l, t)$, and $F_\nu^l(t) = F(x_\nu^l, t)$. (By the *interior* of a refinement, we mean all its points except the r leftmost and q rightmost ones.) The coefficients A_j are assumed to depend smoothly on their arguments. For example, the interior Lax-Wendroff approximation to our model problem (with $t = t_m^l \equiv mk_l$) is

$$v_j^l(t+k_l) = (I - ck_l D_0^l + \tfrac{1}{2}c^2 k_l^2 D_+^l D_-^l)v_j^l(t).$$

The restriction to two-level schemes is necessary to simplify manipulations with refinements. (When the spatial mesh is adjusted at time

t^i, $i = 0, 1, \ldots, s-1$, it would be awkward, and require more storage, to adjust the mesh at previous time levels too.) This also simplifies error estimation, as will be seen in Section 3. (This restriction does not exclude two-level schemes with fractional time steps, such as two-step Lax-Wendroff.)

The restriction to explicit schemes is more fundamental. This is no restriction for purely hyperbolic problems, but can be a restriction for more complicated problems (*e.g.*, coupled heat and sound). Since our algorithm calculates solutions at a given time level piecewise in various parts of the interval $a \le x \le b$, this restriction is not merely for convenience.

In Section 3 we will also require that the local truncation error (per unit time step) of the interior approximation must have the same order in space and time. Since we will most often use interior approximations which are second order in space and time, this restriction is not too severe.

At coarse-fine interfaces (between level $l+1$ and level l refinements) which do not abut boundaries, we use a hybrid method, the *coarse/fine approximation* [9] corresponding to difference method (2.11). We apply the right side of (2.11) to solution values on level l with space step h_l and time step ik_{l+1}, for $i = 1, 2, \cdots, M$ (that is, replace λ_l by ik_{l+1}/h_l), and store the solution value on the level $l+1$ refinement.

For Lax-Wendroff applied to our model problem this is

$$v_{Nj}^{l+1}(t_m^l + ik_{l+1}) = (I - cik_{l+1}D_0^l + \tfrac{1}{2}c^2 i^2 k_{l+1}^2 D_+^l D_-^l)v_j^l(t_m^l).$$

Fig. 2 shows the stencil at the interface between a refinements on levels 2 and 3 ($l = 2$) when $N = M = 3$, so that the third level space step $h_3 = h_2/3$ and the third level time step $k_3 = k_2/3$. When $i = 1$, points A, B, and C are used to advance to point D. When $i = 2$, points A, B, and C are again used to advance to point E, and when $i = 3$, the same points are used to advance to point F.

For a difference scheme (2.11) whose stencil is more than three mesh points wide, we will need to use the coarse/fine approximation r times in Fig. 2 (and correspondingly q times at the right end of a refinement). This is done, *e.g.*, for $r = 2$, by using the stencil illustrated in Fig. 2 to get point D, then shifting the stencil one finer mesh point to the right to get the point to the right of D. (This involves a spatial interpolation in the coarser mesh, which is done as in the mesh-adjustment step).

Finally, boundaries are treated the same as with a uniform mesh; if an l-th level refinement abuts the left boundary, we set

$$v_\mu^l(t+k_l) = \sum_{\sigma=-1}^{0} S_\sigma^{(\mu)} v_r^l(t-\sigma k_l) + g_\mu^l(t), \qquad \mu = 0,1,\ldots,r-1, \tag{2.12}$$

where

$$S_\sigma^{(\mu)}(l) = \sum_{j=-\tilde{r}}^{\tilde{q}} C_{j,\sigma}^{(\mu)}(x_r^l + jh_l,\, t-\sigma k_l,\, h_l)E^j, \qquad \sigma = 0,-1$$

$t = t_m^l \equiv mk_l$, $\tilde{r} \le r$, and $C_{-r+\mu,-1}^{(\mu)} = 0$. The approximation at the right boundary is analogous. For our model problem we use the prescribed values

$$v_j^l(t+k_l) = g(-c(t+k_l))$$

at the left boundary; and upwind differencing

$$v_j^l(t+k_l) = (I - ck_l D_-^l)v_j^l(t)$$

at the right boundary.

Once again we have restricted ourselves to two time levels, for the same reasons as before. We allow ourselves implicit boundary conditions here ($S_{-1}^{(\mu)} \ne 0$) since we can first solve for the points on the right hand side of (2.12) using the explicit interior approximation.

An extremely important feature of this method is the use of a buffer on either end of any refinement (except the coarsest one). If we are estimating the truncation error for the refinement $\{x_j^l\}$ and the error tolerance is exceeded between $j = \alpha$ and $j = \omega$, then we instead refine from $j = \alpha - b_l$ to $j = \omega + b_l$, where b_l is the buffer length for refinements of level $l+1$. That is, both ends of the $l+1$-st level refinement are padded with b_l extra cells of width h_l. In general, if our l-th level refinement requires several intervals of $l+1$-st level refinement (according to the error estimate), then each such interval is padded as above. (This may cause some $l+1$-st level refinements to merge.)

How do we choose b_l? From Fig. 2, we see that b_l should be at least one, because we use the coarse/fine approximation. For safety we make it two. We shall assume that the (given) maximum wave speed is c. For simplicity we shall assume that all partitions P_l, $l \geq 1$, are the same, and that we check the error every ϑ coarse time steps. Therefore, in time k_1 a wave could travel left or right a distance of $c\vartheta k_1 = c\vartheta\lambda_1 h_1 = c\vartheta\lambda_1 N^{l-1} h_l$, or $c\vartheta\lambda_1 N^{l-1}$ cells of width h_l. (Here the $l-1$ is an exponent, not a superscript.) So we take $b_l = 2 + \lceil c\vartheta\lambda_1 N^{l-1}\rceil$, where $\lceil x \rceil$ is the ceiling function (the least integer greater than or equal to x). (For difference approximation (2.11), b_l must be modified by replacing 2 by $q+1$ at the left end of a refinement, and by $r+1$ at the right end.) Obviously, higher level refinements have larger buffers.

The buffer mechanism has several beneficial consequences. First, and most important, it insures that the rapidly varying part of the solution does not escape into the coarser region. As we saw in Section 1, this is absolutely essential to the success of the algorithm. Secondly, this policy allows us to use difference approximations at coarse/fine interfaces which would otherwise not be accurate enough in the fine mesh. We can also estimate the local truncation error at coarse/fine interfaces in a very simple manner (see Section 3.2). Third, it allows "smooth" transitions in mesh width. That is, a level l refinement can abut a level $l+1$ or $l-1$ refinement, but not others. This is important when using recursive refinements. Fourth, it keeps the refinements from splitting into tiny pieces, because level $l+1$ level refinements which are closer than $2b_l$ level l cells apart (before buffering) are joined together. (If the local truncation error were large in absolute value but suddenly changed sign, this might cause splitting into pieces.) We make this condition even more stringent by joining together any level $l+1$ refinements which are less than $2b_l+2$ level l cells (of length h_l) apart (before buffering). Fifth, buffering allows us to specify *a priori* the times to check the local error (and adjust the mesh). In particular, we need not check the error at *every* (fine) time step (Section 8 shows that this is very expensive). We can instead check at every *coarse* time step, or even every ϑ coarse time steps, where ϑ is a small positive integer. Sixth, buffering contributes greatly to the robustness of the algorithm. Buffers make the algorithm relatively insensitive to small inaccuracies in the local error estimation.

Let us comment on the storage required for this algorithm. If we use a two-level method and perform the operations in the order given, then we need two levels of solution values, just as for a uniform mesh. As soon as all the solution values in a refinement at a new time level are known, we can overwrite them on the old solution values. A slight amount of additional storage is needed for pointers and indices; this is minuscule compared to space for solution values. Next, free space is needed to separate the solution values on refinements. The amount is variable, but can be chosen quite small (see Section 6.3). Finally, storage is needed for error estimates; but these can be done a refinement at a time, so we only need two vectors (when solving a scalar equation), each the size of the largest refinement. We did not implement our algorithm in a way which minimizes the amount of storage.

3. Estimation of the Local Truncation Error. In this section we show how to estimate the local truncation error. We examine two methods for estimation in the interior of refinements: differences and Richardson extrapolation. Then we propose methods for estimating the error at coarse/fine interfaces and at boundaries.

The first important conclusion of this section (and of the computations in Section 6) is that several methods can be used to estimate the error, both in the interior and at boundaries. (In [5] we examined two other methods, neither of which is as useful.) Thus our algorithm is quite general as well as robust. The second important finding is that the Richardson method is more convenient in the interior, and differences are more convenient at boundaries.

For simplicity, we shall write all approximations as occurring on a *uniform* mesh. We will always assume that $\lambda = k/h$ = constant, and that the solution of the differential equation has sufficiently many derivatives. When we speak of asymptotic estimates and leading terms, we shall always mean "as $h \to 0$".

3.1. Two Methods. We now explain two methods for estimating the leading terms of the asymptotic expansion of the local truncation error. These methods require extra computations, in addition to those required to compute the solution. We first examine the interior of a refinement.

3.1.1. Differences. We will illustrate this method on an example. The local truncation error for the Lax-Wendroff approximation to problem PI on a uniform mesh with stencil centered at (x,t) is

$$\frac{k}{6}(-k^2u_{ttt}(x,t) - h^2u_{xxx}(x,t)) + O(h^4). \tag{3.1}$$

We use the differential equation to replace t derivatives by x derivatives in this expression. We then obtain

$$ck\frac{h^2}{6}u_{xxx}(x,t)(c^2\lambda^2 - 1) + O(h^4).$$

We may now approximate u_{xxx} by a five-point divided difference at points in the interior of a refinement. Specifically, with t dependence omitted,

$$u_{xxx}(x) \approx (-2u(x-2h) + 4u(x-h) - 4u(x+h) + 2u(x+2h))/4h^3.$$

3.1.2. Richardson Extrapolation. We shall apply this method (which was suggested by J. Oliger) to our linear hyperbolic operator (2.1). In the interior of a refinement we approximate this system by any linear multi-(time) level explicit difference scheme whose local truncation error per unit time step has the same order p in space and time:

$$v_\nu(t+k) = Q(h)v_\nu(t) \equiv \sum_{\sigma=0}^{\rho} Q_\sigma v_\nu(t-\sigma k) + kF_\nu(t),\ \nu = r, r+1, \ldots, N_0-q. \quad (3.2)$$

Here $x = x_\nu \equiv a + \nu h,\ t = t_m \equiv mk$,

$$Q_\sigma = \sum_{j=-r}^{q} A_{j\sigma}(x_\nu+jh,\ t-\sigma k,\ h)E^j \qquad \sigma = 0, 1, \ldots, \rho, \quad (3.3)$$

the $A_{j\sigma}$ are matrix coefficients, and E is the spatial shift operator.

The local truncation error of method (3.2) applied to (2.1) at the point $(x, t) = (x_\nu, t_m)$ is given by

$$\begin{aligned} u(x,t+k) = \sum_{\sigma=0}^{\rho} Q_\sigma u(x,t-\sigma k) + kF(x,t) + k(h^p T_1(x,t) + k^p U_1(x,t)) \\ + O(h^{p+2}), \end{aligned} \quad (3.4a)$$

where T_1 and U_1 are sufficiently smooth functions of x and t. Symbolically,

$$u(x,t+k) = Q(h)u(x,t) + kF(x,t) + k\tau(x,t). \quad (3.4b)$$

For the global truncation error $e(x,t) = u(x,t) - v_\nu(t)$ we obtain the expansion

$$e(x,t+k) = Q(h)e(x,t) + k\tau(x,t) \quad (3.5)$$

by subtracting (3.2) from (3.4). We shall assume that the difference approximation is defined for *all* (x,t), not only at one set of mesh points.

To use the Richardson method, we take one (time) step of the approximation (3.2) with $x = x_\nu,\ t = t_m$, using mesh spacing h in space and k in time. (See Fig. 3 for a two-level scheme with $q = r = 1$, with stencils shifted horizontally for clarity.) We then repeat the step using (3.2) with t_m replaced by t_{m+1} and the same mesh spacing, obtaining the approximation $v_\nu(t+2k)$. (Before performing this second step, we will need to generate more points on level $t = (m+1)k$ by applying (3.2) with (x_ν, t_m) replaced by $(x_{\nu+j}, t_m)$, $j = -r, -r+1, \ldots, 0, 1, \ldots, q$. Thus in practice this estimation is done for all interior points of a refinement at once.) Then we start over at $(x,t) = (x_\nu, t_m)$ using (3.2), but with stencil spacing $2h$ and $2k$ (*i.e.*, replace jh by $2jh$ and k by $2k$ in (3.2) and (3.3)), obtaining the approximation $v_\nu^{(2)}(t+2k)$. We then subtract the two approximations obtained at level t_{m+2}, and divide by $2^{p+1}-2$ to obtain the desired estimate.

We shall prove the validity of this method when both the differential and difference equations have constant coefficients (they may not depend on x or t but those of the difference approximation may depend on h when B in (2.1) is nonzero). However, our numerical results in Section 8 will show that this procedure is applicable in much more general circumstances. In practice, our interior difference approximation will always be two-level ($\rho = 0$). Here we do *not* rewrite t derivatives in the local truncation error in terms of x derivatives.

THEOREM 1. *Approximate the hyperbolic operator* (2.1) *for the Cauchy problem by the consistent multilevel explicit interior difference scheme* (3.2). *Assume that both operators have constant coefficients. If the undifferentiated term Bu in* (2.1) *is nonzero, assume that the coefficients* (3.3) *in the difference operator are smooth functions of h. Assume that the local truncation error per unit time step τ* (3.4) *has the same order p in space and time, that the solution u of the differential equation and the global error $e = u - v$ are sufficiently smooth functions of x and t, and that $\lambda = k/h$ = constant. Then we can estimate the (lowest terms of the asymptotic expansion of the) local truncation error $k\tau(x,t)$ at the point $(x,t) = (x_\nu, t_m)$ in the interior of a refinement using the Richardson method, and*

$$
\begin{aligned}
k\tau(x,t) &\equiv k(h^p T_1(x,t) + k^p U_1(x,t)) + O(h^{p+2}) \\
&= (v_\nu(t+2k) - v_\nu^{(2)}(t+2k))/(2^{p+1}-2) + O(h^{p+2}),
\end{aligned}
$$

where T_1 and U_1 are sufficiently smooth functions of x and t, $v_\nu(t+2k)$ is the approximation obtained by applying one step of (3.2) *with t replaced by t_{m+1} and mesh spacing h and k, and $v_\nu^{(2)}(t+2k)$ is the approximation obtained by applying one step of* (3.2) *with $(x,t) = (x_\nu, t_m)$ and mesh spacing $2h$ and $2k$.*

Proof. The local truncation error $e^{(2)}(x,t+2k) = u(x,t+2k) - v_\nu^{(2)}(t+2k)$ of our double size step is, to leading order terms, 2^{p+1} times the local truncation error for a single step. That is,

$$e^{(2)}(x,t+2k) = Q(2h)e(x,t) + 2^{p+1}k\tau(x,t) + O(h^{p+2}), \tag{3.6}$$

obtained by replacing h by $2h$ and k by $2k$ in (3.5). (From this formula and (3.3) we can see that using a scheme with more than two time levels will entail storing many previous time levels of the solution. This would be highly impractical in multidimensional problems)

We apply two single steps of the method to the error $e(x, t)$. Thus we replace t by $t + k$ in (3.5) and substitute (3.5) into the result. This yields

$$
\begin{aligned}
e(x,t+2k) &= Q^2(h)e(x,t) + k\tau(x,t+k) + Q(h)k\tau(x,t) \\
&= Q^2(h)e(x,t) + k\tau(x,t) + (I + O(h))k\tau(x,t) + O(h^{p+2}) \\
&= Q^2(h)e(x,t) + 2k\tau(x,t) + O(h^{p+2}).
\end{aligned}
$$

Here we have expanded the first τ in a Taylor series about (x,t) and have expanded Q using the consistency. (The consistency relations express the coefficients of the power series in Q in terms of the coefficients of the differential equation, hence the former are indeed bounded.) We subtract this equation from (3.6) to obtain the computable quantity

$$
\begin{aligned}
&v_\nu(t+2k) - v_\nu^{(2)}(x,t+2k) = e^{(2)}(x,t+2k) - e(x,t+2k) \\
&= [Q(2h) - Q^2(h)]e(x,t) + (2^{p+1}-2)k\tau(x,t) + O(h^{p+2}).
\end{aligned}
$$

This will yield our result if we can show that the bracketed expression is $O(h^{p+2})$. Since $Q(h) = I + O(h)$ it is clear that $Q^2(h)$ and $Q(2h)$ agree up through and including first order terms in their formal power series expansions. So we must show that the terms with coefficients h^2 in the bracketed expression are $O(h^{p+2})$. Now e is the same order $O(h^p)$ as τ for the Cauchy problem, since we have assumed smoothness of solutions and error. Hence the terms in question are $O(h^2)$ times derivatives of e, hence $O(h^{p+2})$. This completes the proof.

We have called this a Richardson extrapolation method, but we are using it in a non-traditional way. Both our method and the traditional approaches (for o.d.e.'s and elliptic p.d.e.'s) improve the accuracy of the approximate solution by estimating the local truncation error. But we use the estimate to decide where to refine; the traditional approaches add the estimate to the approximate solution. (Doing the latter would not be useful to us, since our estimation is not being done at every time step.) As a consequence, the traditional approaches improve the order of accuracy of the basic difference scheme; our method does not.

In both error estimation methods, the quantity we control in the interior is not the local truncation error, but the local truncation error per unit time step

$$|\tau(x,t)|_\infty \leq \delta,$$

where δ is the local error tolerance, and $|\cdot|_\infty$ denotes the maximum absolute value of all components of an n-vector at the position (x, t). Thi s because one power of h in accuracy is lost in going from the local truncation error of the interior approximation to the global error [21].

Our computations in Section 8.6 of [5] show that for our model problem P1, either of our methods produces approximately equally accurate estimates of the local truncation error in the interior. And clearly, Richardson estimates are more expensive to compute than difference estimates. However, the Richardson method is considerably more convenient to use in the interior of refinements. The difference method requires us to explicitly compute the local truncation error, and then to replace t derivatives by x derivatives. Even with a symbol-manipulation program like MACSYMA, this can be exceedingly cumbersome for realistic coupled systems. The Richardson method makes possible an (almost) automated approach to local error estimation. One need only know the order p of the method and the factor $2^{p+1}-2$ used to divide the difference $v_\nu - v_\nu^{(2)}$ of the two approximations at time $t + 2k$.

3.2. Coarse/Fine Interfaces. Let us now discuss the modifications needed for coarse/fine interfaces which do not abut boundaries. For concreteness, assume that an l-th level refinement R_l has a descendant $l+1$-st level refinement R_{l+1} which does not abut the left or right boundaries $x = a$ or $x = b$. This introduces two coarse/fine interfaces, namely, the ends of R_{l+1}. (See Fig. 2 for the left end of R_{l+1}.) Recall that, the last time we estimated the error, we added enough padding or buffering (see Section 2.6) to both ends of R_{l+1} to ensure that waves could not escape it, plus two extra level l (spatial) cells. This guarantees that we will not need to refine the ends of refinement R_{l+1} (unless they abut boundaries) and assures "smooth" mesh transitions. Since our local truncation error estimates are used only to decide where to refine, we can safely set our estimate at the ends of R_{l+1} to zero.

The next question is the choice of estimator at mesh points which are one (spatial) mesh point on the "fine" side of a coarse/fine interface, or one mesh point away from a boundary. (This is for the case of a stencil with three adjacent spatial points, *i.e.*, $q = r = 1$ in (2.11) or (3.2). In the case where q or r is greater than one, similar considerations apply to the q or r points on the "fine" side of the interface.) Fig. 3 shows that the Richardson method does not yield an estimate here. We also set this estimate to zero, for the same reasons as before.

3.3. Boundaries. Let us consider local error estimation at boundaries. On the left boundary, there are J boundary conditions specified for the differential equations (2.1)-(2.4). We can approximate these in the obvious way with no local truncation error. We will call these "exact" boundary approximations.

When $r \geq 1$ in the interior difference approximation (2.11) or (3.2), then we need $n-J$ "extra" boundary conditions at the left boundary,

$$v_\mu(t+k) = \sum_{\sigma=-1}^{\rho} S_\sigma^{(\mu)} v_r(t-\sigma k) + g_\mu(t), \qquad \mu = 0, \tag{3.7}$$

where $S_\sigma^{(\mu)}$ is as given in (2.12), but with the appropriate time level. If $r > 1$, we also need $n(r-1)$ additional boundary conditions of type (3.7) for $\mu = 1, \dots, r-1$. (Similar statements hold at the right boundary, with J replaced by $n-J$ and r replaced by q. We will only discuss the left boundary; the right is similar.) We will first consider the extra conditions; at the end of the next subsection we shall examine the "exact" boundary approximations.

The local truncation error $k\tilde{\tau}$ of (3.7) is

$$u(x_\mu, t+k) = \sum_{\sigma=-1}^{\rho} S_\sigma^{(\mu)} u(x_r, t-\sigma k) + g_\mu(t) + k\tilde{\tau}(x,t), \quad \mu = 0, 1, \dots, r-1.$$

For a restricted class of boundary approximations, it is tempting to recycle Theorem 1, using boundary approximations in place of interior operators. Unfortunately, this fails because the boundary operator was not the only operator used to produce solution values at previous time levels. (For example, if we use the first order upwind boundary approximation and the Lax-Wendroff interior approximation on our problem P1 (the first order wave equation), then we apply the former three times and the latter once to obtain a boundary estimate.)

Thus, the Richardson method is not very useful at boundaries for several reasons. First, we must do a complete error analysis of the Richardson method with both boundary and interior stencils for each individual problem. (We did this for problems P1 and P2; see Section 8.7.) This cannot be done in an automated way. Second, there are relatively few boundary approximations which have the same order spatial and time error. Third, this method does not work for implicit approximations.

So we must use differences. In contrast to the interior approximation, it is usually practicable to write down the local truncation error for the boundary approximation, and rewrite t derivatives in terms of x derivatives. For example, in our first order wave equation, if we use upwind differencing at the right boundary

$$v_\nu(t+k) = v_\nu(t) - c\lambda(v_\nu(t) - v_{\nu-1}(t)),$$

the local truncation error is

$$\tfrac{1}{2}\cdot(k^2 u_{tt} - c\lambda h^2 u_{xx}) + 0(h^3);$$

replacing t derivatives by x derivatives yields

$$\tfrac{1}{2}ckh(c\lambda-1)u_{xx} + O(h^3).$$

We then replace the u_{xx} term by a three-point one-sided difference.

The local truncation error for extrapolation boundary conditions

$$(hD_+)^j v_0(t+k) = 0, \qquad \text{for fixed } j \geq 1$$

can only be estimated by differences. As an example, for $j = 2$, we estimate the truncation error

$$h^2 u_{xx} + O(h^3)$$

by using a four-point one-sided difference (since the three-point estimate yields zero).

In Section 8.7 we numerically compare different methods of error estimation at boundaries.

Gustafsson's [21] analysis and our computations in Section 8.5 show that the order of accuracy of the boundary approximation may be one order lower (but not less) than that of the interior approximation, in order to preserve the global order of accuracy, which is then the same as the order of the local error per unit time step for the interior approximation. This means that when we are deciding whether to refine at the boundary, we should *not* control the local error per unit time step, but instead the local error,

$$|k\tilde{\tau}(x,t)|_\infty \le \delta.$$

3.4. Boundary Anomalies. We will now discuss an anomaly which arises frequently in model problems, but quite seldom in realistic problems. This problem can only arise with (a) a single scalar equation; (b) an $n \times n$ system which can be decoupled into independent subsystems; or (c) a system with the property that all the characteristic variables at a boundary are inflow variables, i.e., all the characteristics point into the region (e.g., supersonic inflow). If this is the left boundary, then for problem (2.1)-(2.4) in diagonal form, the boundary condition (2.3) has $S_{\mathrm{II}} = 0$ (since u^{II} has no components, *i.e.*, $u = u^{\mathrm{I}}$).

This problem arises because no coupling occurs between inflow and outflow variables. Using the obvious difference approximation to these inflow boundary conditions in the differential equation (the so-called "exact" boundary approximations) yields zero local truncation error.

Examining our problem P1, we can see a possible difficulty with using zero in our error estimation. The left boundary condition contains a forcing (inhomogeneous) term g which results in the wave entering the region. Clearly, we want to put refinements around any "large" wave entering the left as soon as possible. If we set the truncation error estimate to zero at the boundary, we will not detect the wave until it has already entered the region. This can be remedied by treating the forcing term g_1 or g_2 of (2.3) or (2.4) as generating a fictitious local truncation error. For our first order wave equation, where $u(0,t) = g(t)$, we write down the local truncation error (3.1) for the interior approximation, then replace x derivatives by t derivatives, using the differential equation. Since $u_{ttt}(0,t) = g_{ttt}$, we can analytically differentiate g to arrive at the result. This procedure was used in our computations with problem P1 (see Section 8.4). (Notice that we used an interior error for a boundary approximation, and then controlled the local error per unit time step. We could equally well have used the boundary error, which depends on u_{tt} and controlled the local error. In either case we would control the same power of h.)

In case (c), we may also proceed as in our first order wave equation. For a local truncation error of the form

$$\alpha u_{xxx} + \beta u_{ttt} \tag{3.8}$$

we rewrite x derivatives in terms of t derivatives using the differential equation,

$$u_x = A^{-1}(u_t - Bu - f),$$

and replace the first term in (3.8). Since $u^1_{ttt} = (g_1)_{ttt}$, we can again analytically differentiate to arrive at the result. In practice, the rewriting may be cumbersome.

In the vast majority of cases, our problem will not have property (a), (b) or (c). Then some of the components of the difference approximation have nonzero local truncation error, and our usual error estimates for these component(s) will detect any incoming wave, since the boundary conditions are coupled. This technique was used in our problem P2 (the second order wave equation) in Section 8.

4. Stability.

In this section we give a brief discussion of stability.

We begin with a definition. The square of the (discrete) $l_2(x)$ *norm* of an n-vector v defined on our refined grid at time t^i, $i = 0, 1, \cdots, s$ is a sum of terms, one for each mesh point existing at the given time. (If a point (x, t) is covered by more than one grid point at different levels, use the finest one.) Each term is of the form $h_i |v_j^i(t^i)|_2^2$, where h_i is the interval to the left of the grid point, and $|\cdot|_2$ denotes the sum of squares of the components of the n-vector. (At the left boundary, use the interval to the right of the mesh point.)

The usual definition of stability for the initial boundary value problem is Definition 3.3 of Gustafsson, Kreiss and Sundström [22] (hereafter referred to as the GKS definition). But there are several problems with this approach, if we wish to use this definition to prove convergence.

The GKS definition assumes that the initial data function f is zero. This may be acceptable for $t = 0$, but after we integrate over the horizontal strip S_1, we in effect start a new initial boundary value problem at $t = t^1$, and now the "initial" data are not zero. It is difficult to incorporate a nonzero f into the GKS definition, since it was derived using Laplace transform. However, it is necessary if we wish to use it to prove convergence. For, in an interval $0 \leq t \leq T$ the number of strips S_i becomes unbounded as $h \to 0$. The solution at $t^s = T$ depends on the values of the solution ("initial data") at all previous strips, but this dependence on values at times t^{s-1}, to $t^{s-2}, \ldots, t^1$, has to be removed if we want to prove convergence. This can only be done if f appears explicitly.

A second difficulty is related to the first. The GKS definition assures us that $\exp(-\alpha t)$ times the solution is in $l_2(x,t)$ (the usual discrete Hilbert space in x and t), but for any fixed t does not assure us that the solution is unconditionally in $l_2(x)$ (the discrete Hilbert space in x). (Compare Theorem 3.1, GKS.) In order to integrate over a new strip S_i, we wish to treat the solution values at $t = t^{i-1}$ as "initial" values, and this requires that they be in $l_2(x)$.

For these reasons, Oliger [4], [36] has proposed a new stability definition. It applies to approximations in any number of space dimensions, and is the discrete analog of the well-posedness condition for differential equations. We state it for a problem in one space dimension.

DEFINITION 2. Let $\lambda = k_l / h_l$ = constant, independent of l. Assume that we adjust the mesh only at coarse time points. The difference approximation (2.10) - (2.12) on our region R is *stable* for a refined mesh (as described in Section 2.2) if for any $T > 0$, there exists a constant $K_T > 0$ such that, for all positive integers s, all sets of time division points (2.6), all $k_l > 0$, $l = 1, 2, \ldots, \Lambda$, satisfying our restrictions for refined meshes, and all F, g_μ, I, and f, an estimate

$$\|v^s(T)\|_x + \sum_{i=1}^{s} \sum_{\mu=0} \|v_\mu\|_{[t^{i-1}, t^i]}$$
$$\leq K_T \left(\|f\|_x + \|F\|_{x,[0,T]} + \sum_{i=1}^{s} \sum_{\mu=0} \|g_\mu\|_{[t^{i-1}, t^i]} + \sum_{i=1}^{s} I(t^{i-1}) \right).$$

holds.

Here we have not defined all our norms, and have altered our notation [5] from Section 2. So we shall explain the meaning of the terms.

The first term on the left is the discrete $l_2(x)$ norm (just defined) of the approximate solution at time $t = T$. The second term is the sum of discrete $l_2(t)$ norms of the approximate solution at the boundaries, one for each strip S_i. The first term on the right is the discrete $l_2(x)$ norm of the initial function. The second term is the discrete $l_2(x, t)$ norm of the inhomogeneous forcing function in (2.11). The third term is the sum of discrete $l_2(t)$ norms of the inhomogeneous boundary terms (2.12), one for each strip S_i. Finally, the fourth term is the sum of errors which arise when we interpolate from a coarse mesh to a finer one during mesh adjustment or creation.

For a uniform mesh, we believe that a scheme is GKS stable if it is stable in the sense of Definition 3.6, either for a quarter-plane or strip problem. We believe that the converse is not true in general; that is, the new definition is stronger.

A necessary part of any stability proof is stability of two quarter-plane problems joined along a coarse/fine interface. This question has been examined (using the GKS definition) in Ciment [9] and Oliger [35] for the case of equal time steps on both sides of the interface. Oliger found that if leap-frog was used on both sides of the interface, certain restrictions on the refinement ratio needed to be made. But if the difference scheme was dissipative on one side of the interface, all stability problems vanished. Berger [2] proved the GKS stability of the Lax-Wendroff method on a model problem along a coarse/fine interface with unequal time steps. For these reasons we have used a dissipative scheme in all our calculations.

5. Error Analysis. This section summarizes partial convergence results which give a theoretical justification for our algorithm.

It is clear from the description of our algorithm (and from the computations in Section 8.5) that mesh refinement does not increase the (global) order of accuracy of a difference approximation (compared to using a similar approximation on a uniform mesh). How then does mesh refinement achieve its efficiencies?

In [5] we gave the answer using a proposition which we did not prove, but which is well-supported by computational evidence (see Section 8). It is the direct analogue of Gustafsson's [21] result on the rate of convergence of difference approximations to the initial boundary value problem for hyperbolic systems in one space dimension. Our proposition states that a result similar to Gustafsson's holds when (a) we replace a uniform grid by our refined grid; and (b) the difference scheme is stable according to Oliger's stability definition rather than the GKS definition. Using the norms given in the previous section, this proposition states that the discrete l_2 norm of the global error at time $t = T$ is bounded by a constant K_T' times a sum of four terms. These terms are the discrete l_2 norms of the interpolation error, and of the local errors of the interior, interface, and boundary approximations.

This proposition answers the question, How does the order of accuracy of the interpolation, and of the interior, boundary and interface approximations affect the global error? In particular, suppose that one uses an $O(h^2)$ approximation in the interior of refinements and at coarse/fine interfaces. Also suppose that one uses $O(h)$ boundary approximations and linear interpolation ($O(h^2)$) to obtain solution values on $l+1$-st level refinements from l-th level refinements. Finally, assume that this scheme is stable in Oliger's sense. Then, subject to certain compatibility and smoothness assumptions, the proposition states that the global error is $O(h^2)$.

Now suppose we use a strategy suggested by de Boor [12] and Pereyra and Sewell [37] in other contexts: arrange our spatial mesh so that it "approximately equidistributed" the hybrid local truncation error at time $t = t^{i-1}$, $i = 1,2, \ldots, s$; compute forward in time until the equidistribution condition is "nearly violated" at time $t = t^i$; and then approximately equidistribute again. (The hybrid truncation error is a suitable blending of the interior, interface and boundary truncation errors.) In practice, of course, it is more work to discover whether the condition is "nearly violated" than it is to simply approximately equidistribute again. That is why we choose our "equidistribution" (mesh adjustment) times *a priori*. A somewhat similar strategy has been used by Gannon [16] for parabolic problems with finite elements in two space dimensions.

We should emphasize that we do not approximately equidistribute in practice, as it would be too expensive. Our recursive refinements achieve a primitive form of approximate equidistribution at much less cost.

Using our proposition and results of Pereyra and Sewell we can then show (generalizing what Oliger [33] did for the Cauchy problem) that, loosely speaking, using our mesh refinement algorithm multiplies the constant K_T' by the factor

$$\mu_{\max}^{2p}\left(1 + \frac{K(q+r)}{(b-a)}\right).$$

Here

$$K = \prod_{\mu=1}^{\Lambda-1} N^{(\mu)}$$

is an upper bound on the ratio $\max_j h_j / \min_j h_j$ of spatial step sizes in our refined mesh; $\mu_{\max}$ is, loosely speaking, the ratio of the length of the spatial region that needs refinement (has high truncation error) to the length of the whole spatial region $b - a$; p is the order of the interior approximation and the global error; and q and r are the stencil sizes given in (2.11). When the solution has rapid variations only in a small part of the (spatial) region, then the local truncation error is small over most of the region, and $\mu_{\max}$ is therefore small. Thus our algorithm can use fewer mesh points in regions where the local truncation error is small (compared to using the same difference scheme on a uniform mesh which achieves the same level of accuracy) and this produces significant economies, as shown in Section 8.4.

6. Data Structures. In this section we discuss the data structures used in our mesh refinement algorithm. The data structure has two parts: a four-way linked tree to show the relationships between refinements, and sequentially allocated deques for storing solution values of refinements. We will describe the deques first.

6.1. Deques. One of the most important operations on a refinement is to "move" it left or right to follow a wave. An efficient way to "move" a refinement right (without actually moving any grid points) is to add grid points to the right and delete them from the left. This leads us to store the solution values for a refinement in a data structure called a "deque", or double-ended queue [28]. A deque has the property that items are added to or deleted from its ends, but are never inserted in or deleted from the middle.

For simplicity, we first consider a scalar equation. A natural way to store a collection of deques is sequentially, as shown in Fig. 4. Here we see a region with two refinements, and one of the refinements itself contains a refinement. The solution values for the coarsest mesh occupy a fixed region at the lowest end of a vector which we will call v. The solution values for refinements occupy contiguous sections of the remaining available memory, with variable-width gaps of free space separating them. The gaps allow us to expand or contract refinements (to a limited degree) without moving the solution values. The solution values corresponding to refinements are ordered as follows.

The coarse mesh is labelled refinement 1. It is followed in v by the "second level" refinements (labelled 2 and 3 in Fig. 4), which are ordered in v in the same order as the refinements are encountered in proceeding from left to right in the computational region. Following these are the "third level" refinements (as is refinement 4 in Fig. 4), again in the same order as they occur in a left-to-right scan of the computational region, and ignoring the positions of any coarser (second level) refinements encountered in the computational region. Then would appear all fourth level refinements, and so forth. This scheme duplicates certain solution values in the vector v, namely the ones which correspond to mesh points which lie on different level refinements. However, doing this makes the program much simpler.

For an $n \times n$ system of equations, we replace the solution value vector v by a matrix with n rows. Each row has the same arrangement of solution values separated by gaps, as illustrated in Fig. 4. For simplicity, in the rest of this section we shall again refer to v as a vector.

6.2. The Tree. Next we will describe the (four-way linked) tree of records which shows the relationships between refinements. Trees are natural here, since we use recursive refinements, and are used in most adaptive solvers for elliptic equations. There is a one-to-one correspondence between nodes (records) of the tree and refinements, with the root corresponding to the coarsest mesh. In the following, we will identify a refinement with its node (record), and use the term "refinement" to mean "the node corresponding to a refinement". We will sometimes call the coarsest mesh a "refinement" for uniformity.

The root of the tree has level 1, its immediate successors are at level 2, their successors have level 3, and so forth. Each node contains all the information about a refinement, except its solution values. We will now describe this information. The indices *base* and *top* indicate where in the vector v the solution values for a refinement are located. This is shown in Fig. 4 for the fourth (level 3) refinement, but omitted for the other refinements to avoid clutter. Also shown is a pointer *coarse* to the parent of each refinement. Furthermore, we need pointers to all refinements ("children") of a refinement. We can avoid using a variable number of pointer fields for this by using the usual device. We use one pointer to the leftmost descendant (called *fine* in Figure 6.1) and then chain together all immediate descendants (children) using the "right" pointers, called *rlink* in the figure. A refinement other than the root also needs two indices to denote its endpoints within its parent, that is, which part of its parent it refines. These are not shown in the figure.

Since we will often add or delete nodes in an unpredictable order, we implemented the tree as a linked list. So far our records form a *triply linked tree*, exactly as in Knuth [28, p.352]. However, additional links are needed.

The solution is advanced in time, and the error is estimated a level at a time. Because we already have the *rlink* pointers, we can chain together *all* refinements on the same level (not just those with a common parent) using *rlink*. Then we introduce a vector of pointers pointing to the leftmost refinement on each level. (These are shown in Fig. 4.) This is related to the *level-order representation* of a tree [28, p.350].

The last operation needed on our data structure is a repacking of the v vector, to be discussed shortly. This requires us to sweep through v in both directions. Thus we also require our *rlink* pointers to point from the rightmost refinement (node) on level l to the leftmost refinement on level $l+1$. To enable a leftward sweep, we introduce "left" pointers *llink*, which are inverse to the *rlink* pointers. That is, if node p has right pointer *rlink* pointing to node q, then q has left pointer *llink* pointing to p.

The final result is a four-way linked tree: a triply linked tree with the additional property that all the nodes are linked together, in level order, in a doubly linked list. Our tree uses an inconsequential amount of memory, compared to the memory in v devoted to solution values.

We now examine how the operations on refinements are effected using this data structure. Advancing the difference approximation (in time) or estimating the error can be done a level at a time, using the *rlink* pointers and the leftmost pointers on each level. Here we also use the "ancestor" or *coarse* pointers to copy solution values from finer meshes to coarser meshes for points x which lie on more than one refinement.

Similarly, we adjust the refinements level by level, starting with the highest (finest) level. The mesh adjustment operations can be effected using four elementary operations, which are natural for a deque. They are shorten left, shorten right, extend left, and extend right. Shortening either end of a refinement is a trivial operation, accomplished by moving a *base* or *top* index. Deleting a refinement is the same, but also involves removing a record from the tree and reclaiming its storage. If there is enough space available, extending either end of a refinement involves changing an index, copying solution values from the parent refinement, and filling in new solution values using linear or quadratic interpolation in space. Creation is the same, plus the operation of inserting a new node in the tree. Separation of a refinement into two refinements involves changing indices and inserting a new node. Finally, merging two refinements is simplified because we insisted on the left-to-right ordering of refinements in the vector v. We move left the solution values of the right refinement, if necessary, then extend the left refinement to the right, change some indices, and delete the right node. Complicating the last two operations is the need to adjust pointers to descendant refinements.

6.3. Memory Repacking. A problem occurs during an "extend" operation when there is insufficient expansion room between refinements. This calls for a repacking of memory, and an algorithm for this is given by Knuth [28, pp.245-6] for the case of a sequence of stacks (rather than deques). We will therefore describe the modifications to this algorithm for our data structure.

When a refinement runs out of room in the v vector, moving only the adjacent refinement will probably cause *another* repacking to occur soon, so it is better to reallocate *all* available memory when a refinement runs out of room. Knuth breaks this into two parts: Algorithm G, which decides how to allocate the free memory to the refinements, and Algorithm R, which actually moves the refinements into the positions dictated by Algorithm G. It is Algorithm R which requires the forward and backward sweep of the v vector in order to avoid overwriting any information.

We used Algorithm R unchanged and modified Algorithm G as follows. Knuth's main idea is to share ten percent of the free memory equally among the refinements, and to divide the other ninety percent proportionate to the amount of increase in refinement size since the previous repacking. This idea is not useful in our case. For a traveling wave, all refinements stay about the same size, but "move". However, we can modify this rule by awarding the ninety percent of available memory proportionate to the amount each refinement has *moved* since the last repacking. We discover whether a refinement has moved primarily left or right (in memory) since the last repacking, and award its share of the ninety percent to its left or right, respectively.

This change to Knuth's algorithm greatly reduces the number of repackings compared to more naive allocation methods. Since the coarse mesh doesn't move it receives none of the ninety percent allocation Furthermore, the higher level refinements move further (measured in number of mesh points, not physical distance) than the lower level ones, so they are awarded more free space by this scheme.

Our storage scheme for solution values cannot be generalized to more space dimensions, because refinements no longer have only two "ends". In more dimensions it is also unwise to merge adjacent refinements, even if they are on the same level. But our scheme avoids more complicated storage allocation algorithms [2].

7. Choice of Programming Language. We will now explain and justify our choice of implementation method for the programs used in our computations.

Possible alternatives include Algol W, PL/I, Algol 60, Algol 68, Pascal, Fortran, and Fortran with preprocessor. The arguments against the first four are lack of availability of a compiler and/or lack of portability. Raw Fortran (even Fortran 77) is cumbersome to use because of the lack of control and data structures, both of which are crucial for our task. However, most numerical software is written in Fortran, and if one uses another language, there must be an interface to Fortran. For reasons given in Kernighan [27], we did not use Pascal, despite its excellent data structuring facilities.

We then examined preprocessors. We rejected Feldman's [15] EFL, Grosse's [20] language T, and Stein's [45] language for portability reasons, even though their respective authors have devoted considerable thought to devising appropriate language constructs for numerical algorithms. We examined two portable Fortran preprocessors: Kernighan's [26] Ratfor and Cook and Shustek's [10] Mortran. Although the former is more widely used, we chose the latter because it is far more general and flexible. (Engquist and Smedsaas [14] have also used a macro processor in their work. Gropp [19] has developed a language for mesh refinement algorithms.)

The term Mortran, like Fortran, has several meanings. It can mean a structured source language, a translator for that language, or a macro-processor. The structured language is implemented as a set of macros which are used by the Mortran macro processor to translate the language into Fortran. The resulting Fortran program is then run like any other Fortran program.

In contrast to most other Fortran preprocessors, the Mortran preprocessor is written in a portable subset of ANSI (standard) Fortran. Hence the Mortran preprocessor, and, more importantly, Mortran source programs, are portable between different machines. (We ran our programs on an IBM 370/168, a CDC 7600, and a DEC VAX with minimal conversion problems.) Furthermore, Mortran source and Fortran source can be intermixed, so the Mortran user has access to all existing Fortran software.

We felt that there was one property of Mortran which made it especially desirable for this project: *extensibility*. This means that new data structures, operations on data structures, and control structures can be added to the language (at rather small cost in implementation time) by adding additional macros to the language. To implement the linked list for our four-way linked tree, we needed records and pointers, and Mortran allows us to create these new data types and to define operations (such as following pointers) on these data types. We modified the Mortran macros for records and pointers given in Zahn [49], and used Pascal-like syntax [25].

Of course, many of the restrictions of Fortran remain. Among these are lack of dynamic storage allocation, arrays with arbitrary subscripts, and recursion. Except for recursion, these are not serious. We needed recursion when advancing the solution and when plotting it; for the latter we needed to search the tree in preorder to produce solution values ordered with increasing x. Doing without recursion made for more obscure code.

For most control constructs, Mortran produces Fortran code that is as efficient as possible without using global flow analysis. For one type of loop (the **for** loop) we rewrote the macros to generate more efficient Fortran [5].

A criticism often levelled at macro preprocessors is that they produce error diagnostics in terms of Fortran instead of the source language. Zahn [49] shows how to write additional macros to produce reasonable source-level diagnostics for the record and pointer constructs. These macros also insure that a pointer points only to a record of the appropriate type.

In [5] we gave a complete program listing of our mesh refinement algorithm applied to problem P2 in Section 8.

8. Computational Results. In this section we answer the following questions about our method:

1. Does our method "follow" or "track" steep gradients? Is it fooled by background effects?
2. Is the algorithm sufficiently general to allow refinements to be created, destroyed, merged, separated, moved, and to abut boundaries?
3. Is the algorithm sensitive to the direction of characteristics, or dependent on knowing that certain boundary conditions are inflow or outflow?
4. Will the method handle nonlinear problems?
5. How well will the method follow discontinuities or shocks?
6. Are recursive refinements worthwhile?
7. How should one choose the refinement ratios N and M?
8. How efficient is the method, both in execution time and memory?
9. How does the global error behave as $h \to 0$?
10. How do the two methods of interior local error estimation of Section 3 compare in accuracy and efficiency?
11. How do different boundary approximations and methods of estimating their error affect the solution?
12. How often should one monitor the local truncation error (and adjust refinements)?

8.1 Model Problems. Since we believe it is impossible to answer these questions analytically, we resort to numerical experiments on model problems. Problem P1, the first-order wave equation, was introduced in Section 2.4. We now introduce two additional problems.

P2 is the second order wave equation, written as a 2×2 first order system, with "open" boundary conditions (*i.e.*, the boundaries are "transparent" to traveling waves). As exact solution we use two counter-streaming Gaussian pulses, superimposed on a sinusoidal background. The differential equation is

$$u_t = Au_x, \qquad a \le x \le b,\ 0 \le t,\ 0 < c, \tag{P1}$$

where

$$A = \begin{bmatrix} 0 & c \\ c & 0 \end{bmatrix},$$

with initial conditions

$$\left.\begin{aligned} u_1(x,0) &= f(x) + g(x) \\ u_2(x,0) &= -f(x) + g(x) \end{aligned}\right\}, \qquad a \le x \le b,$$

and open boundary conditions

$$\left.\begin{aligned} u_1(a,t) &= u_2(a,t) + 2f(a - ct) \\ u_1(b,t) &= -u_2(b,t) + 2g(b + ct) \end{aligned}\right\}, \qquad t \ge 0.$$

We choose $a = 0$, $b = 4$, $c = 1$. The exact solution is

$$u_1(x,t) = f(x - ct) + g(x + ct),$$
$$u_2(x,t) = -f(x - ct) + g(x + ct).$$

To produce our interacting pulses, we take $f(x)$ exactly as in P1, and

$$g(x) = -\exp(-\alpha(x - 4.5)^2),$$

where $\alpha = 200$. Each pulse occupies about 8 percent of the region $a \le x \le b$.

The difference approximation in the interior is again Lax-Wendroff

$$v_j(t + k_l) = (I + Ak_l D_0^l + \tfrac{1}{2}A^2 k_l^2 D_+^l D_-^l)v_j(t)$$

(omitting superscripts l on v), with coarse/fine approximation

$$v_{Nj}^l(t + k_l) = (I + Ak_l D_0^{l-1} + \tfrac{1}{2}A^2 k_l^2 D_+^{l-1} D_-^{l-1})v_j^{l-1}(t)$$

at interfaces, the obvious initial condition, and obvious boundary approximations for v_1. For v_2, we need extra boundary conditions at both $x = a$ and $x = b$, and we use either

(a) upwind/downwind differencing:

$$v_{j2}(t+k_l) = (I + ck_l D_+^l)v_{j2}(t) \quad \text{at } x = a,$$

$$v_{j2}(t+k_l) = (I + ck_l D_-^l)v_{j2}(t) \quad \text{at } x = b,$$

where $v_{j2}(t)$ denotes an approximation to $u_2(x,t)$ at $x = a + jh_l$ on an l-th level refinement; or

(b) first-order extrapolation

$$(D_+^l)^2 v_{j2}(t+k_l) = 0 \quad \text{at } x = a,$$

$$(D_-^l)^2 v_{j2}(t+k_l) = 0 \quad \text{at } x = b.$$

(The first 2 in each line is an exponent, not a superscript.) Gustafsson, Kreiss and Sundström [22] showed that both approximations (a) and (b) are stable with Lax-Wendroff, according to their stability definition.

Our second problem is the Riemann shock tube problem for a compressible, inviscid gas [43]. The compressible Euler equations, in conservation form, are

$$\mathbf{U}_t + \mathbf{F}(\mathbf{U})_x = 0, \qquad 0 \le x \le 1,\ t \ge 0,$$

where

$$\mathbf{U} = \begin{bmatrix} \rho \\ m \\ e \end{bmatrix}, \quad \text{and} \quad \mathbf{F}(\mathbf{U}) = \begin{bmatrix} m \\ (m^2/\rho) + p \\ (e + p)m/\rho \end{bmatrix}.$$

Here ρ, m, and e are the mass, momentum, and energy, respectively, per unit length, and p is the pressure. Let $u = m/\rho$ be the velocity. If the gas is polytropic with ratio of specific heats γ, we can express the energy as $e = \rho(\varepsilon + \frac{1}{2}u^2)$, where the internal energy per unit mass $\varepsilon = p/(\gamma - 1)\rho$. We can also write $p = (\gamma - 1)(e - \frac{1}{2}m^2/\rho)$.

Initially we will have two gases at rest, but at different pressures and densities, separated by a membrane. At time $t = 0$ the membrane is instantaneously removed. Following Sod [43], let us define the ***left state*** $S_l = (1.0, 0., 2.5)^T$ and the ***right state*** $S_r = (.125, 0., .25)^T$. Then, initially, $\mathbf{U}(x, 0) = S_l$ for $0 \le x < 0.5$, and $\mathbf{U}(x, 0) = S_r$ for $0.5 \le x \le 1$. At the left boundary $\mathbf{U}(0, t) = S_l$, and at the right boundary $\mathbf{U}(1, t) = S_r$, both for $t \ge 0$, but before reflections off the wall. (One should actually specify only one of ρ, m, or e at each boundary, as can be seen by considering characteristics and Riemann invariants. However, our procedure is equivalent before reflections off boundaries, since the solutions are constant near the walls.) For fixed $t > 0$, before reflections off the walls, the exact solution is a function of $(x - 0.5)/t$ only. Proceeding from left to right across the region, the exact solution is the constant state S_l, followed by a rarefaction, followed by a constant state. To the right of this is a contact discontinuity, followed by another constant state, then a shock, concluded by the constant state S_r. Sod shows how to compute the exact solution at any time by solving a 3 by 3 system of nonlinear equations.

We approximated this using the usual two-step Lax-Wendroff method, as given in Richtmyer and Morton [41, p.302]. We added dissipation by subtracting from the flux $F_{j+1/2}^{n+1/2}$ occurring in the two-step Lax Wendroff scheme the quantity $\nu|u_{j+1}^n - u_j^n|(\mathbf{U}_{j+1}^n - \mathbf{U}_j^n)$, with $\nu = 1$. That is, the dissipation is based entirely on the values of $\mathbf{U}$ and u at the *old* time level. We chose this method rather than Sod's because it is easier to implement. At the initial jump discontinuity we took average values for the density and energy. Naturally we modified all this by adding subscripts and superscripts l (for level of refinement) in the appropriate places. Since our previous calculations used time-dependent boundary conditions, we use constant ones here. The solution was calculated up to and including time $t = 0.15$. (Of course, at coarse/fine interfaces, we made modifications as in P1 and P2.)

8.2 Qualitative Results. Figs. 5(a) through 5(o) show our algorithm applied to the second-order wave equation with counter-streaming pulses. We used a coarse mesh of 81 points, with refinement ratios $N = M = 3$ and maximum number of refinement levels = 5. (Only four levels were actually used in the calculation.) The local error tolerance was 0.01, $\lambda_1 = 0.8$, and the wave speed $c = 1$. We used the Richardson method to estimate the error.

Each graph plots the approximate solution component $v_i(x,t)$, $i = 1, 2$, versus x (at a fixed time) as a solid line, together with a separate calculation done with no refinement (maximum refinement level = 1), shown as a dotted line. We usually show the first component $v_1(x,t)$ versus x, but in a few instances we show $v_2(x,t)$ versus x. Since the maximum error in this calculation was less than 2 percent, the refined solution may be taken as the exact solution on the graph.

In Fig. 5(a) the pulses have not yet entered the region and we see only the sinusoidal background. (This graph was made at $t = 0.04$ rather than $t = 0$ since we used the exact solution at $t = 0.04$ to compare with another method not given here. We started at $t = 0$ in problem P3.) In Fig. 5(b) both pulses have entered and refinements on levels 2, 3 and 4 have been created at both boundaries. (The small numbers at the top are the level numbers of the ends of refinements.) All the refinements abut the boundaries. In Fig. 5(c) both pulses have left the boundaries and are moving towards each other. The refinements follow.

In Fig. 5(d) the second-level refinements are about to merge, and in Fig. 5(e) they have merged; however, the third and fourth level refinements have not yet merged. Note that the unrefined solution has become a very poor approximation to the pulses--the unrefined peak and trough have only half the size they should. Note also that behind the pulses, the unrefined solution has a large undershoot. In Fig. 5(f) the third level refinements have merged. Fig. 5(g) shows the second component of the solution at the same time. In Fig. 5(h) the fourth level refinements have merged. Fig. 5(i) shows the second component of the solution at the same time.

In Fig. 5(j) the pulses have crossed, and the third and fourth level refinements have separated. (Note that the pulses cross, but the refinements do not.) In Fig. 5(k) the second level refinements have separated as well. Note the degradation in the unrefined solution at this point. Fig. 5(l) shows the pulses approaching the boundaries. Now the unrefined solution has phase errors as well as amplitude errors. In Fig. 5(m) the pulses are leaving the region, and he refinements again abut the boundary. Fig. 5(n) shows that the fourth level refinements have been deleted. Finally, Fig. 5(o) shows that all refinements have been deleted. Only the sinusoidal background remains.

This problem answers questions 1 to 3. It shows the interactions of up to seven refiı.ements. Since both boundaries act as inflow boundaries at some times and as outflow boundaries later, our method does not depend on the direction of characteristics. These calculations also show that the method is not fooled by background oscillations, and that the refinements "follow" and resolve steep gradients. The method clearly adapts to time-dependent boundary conditions.

Figs. 6(a) to 6(d) show the algorithm applied to the Riemann shock tube problem (P3). Shown at time $t = 0.15$ are the density, velocity, pressure and internal energy *vs.* x. The dotted line shows the unrefined calculation. We used 101 points on the coarsest mesh, with refinement ratios $N = M = 4$, and maximum number of levels = 3. The local error tolerance was 0.01, and the maximum wave speed input to the program was $\max(|u| + c) \approx 2.2$. We took $\lambda_1 = 15/37 \approx .405405$, since $\lambda = 5/11 \approx .45454$ is necessary for CFL stability in the absence of dissipation. The maximum number of refinment levels was 3. We used the Richardson method. (In contrast to the smooth solution case, the local truncation error estimate around a discontinuity does not decrease on finer meshes; hence our method will always use the maximum number of levels in this case.) The purpose of this calculation is to see if our method can handle a non-linear problem, and if it can "follow" and resolve shocks, contact discontinuities, and rarefactions, for which it was not designed. These results are directly comparable with those of Sod [43].

The shock, rarefaction and sonic point (at the right side of the rarefaction) are very well resolved. The contact discontinuity has wiggles, since no dissipation was added around it. The maximum error occurs here also. In this problem most of the error comes from the initial step function, and our method "instantly" refines around the initial step. The method has placed the third level refinements only where needed -- around the shock, contact, and the edges of the rarefaction.

This problem shows that the Richardson error estimation method performs correctly under far more general circumstances than the hypotheses of Theorem 1. We need not assume constant coefficients, linearity, or a Cauchy problem. This method performs well even when the global error has a lower order than the local error, and when the solution is not continuous. (It is well-known [31] that the global error is not second order in those parts of the region reachable by characteristics emanating from the shock.) In fact, for smooth solutions this method yields estimates with approximately 10 to 15 percent error. (One of the methods examined in [5] gave 800 to 1000 percent errors!)

Of course, we cannot yet seriously suggest this as a method for computing shocks, since we have not considered matters such as entropy, monotonicity, or conservation. (But see [2] for a discussion of conservation.) This problem is too simple a model for problems with discontinuous solutions, since there are no collisions of shocks and/or rarefactions. For some problems (e.g., chemical kinetics) it is important to accurately resolve shocks. For others (e.g., gas dynamics) our method (using its present error estimation) may expend too much effort resolving shocks. In the latter case a method such as that of Woodward and Colella [48] may be more appropriate. One could also construct a hybrid scheme, by applying a difference method designed for shocks only on high level refinements containing the shock(s), and using a more inexpensive difference method on the rest of the refinements.

We now proceed to more quantitative questions.

8.3 Choosing Refinement Ratios and Maximum Level. In [5] we studied the questions of how to choose the refinement ratios N and M and whether to use recursive refinements with computations on model problem P1. We found that:

(a) It is necessary to use recursive refinements for efficiency.

(b) If we fix the coarse mesh size and the local error tolerance, and let the maximum number of refinement levels increase, then, for smooth solutions, the number of refinement levels used first increases and then stays constant. That is, the method only uses as many levels as are necessary. For smooth solutions, one should therefore set the maximum refinement level at some large number. (In many cases, the method uses three levels.)

(c) One can choose the refinement ratios N and M to be between 4 and 6 for near optimal efficiency. Nothing is gained by choosing different refinement ratios for different l, or by choosing $N \neq M$.

We next come to the most important result of this paper.

8.4 Efficiency of the Method. In Section 8.2 we showed that our method was able to resolve steep gr dients, and even shocks, in the solution. We showed this by comparing the solution obtained by our method with one obtained on a uniform coarse mesh. This clearly showed the qualitative superiority of our "refined" solution over the "unrefined" one.

However, the "unrefined" solution cost far less to compute than the "refined" one. For example, it cost 1.17 seconds to compute a "refined" solution of Problem P1 up to time $t = 3.6$ (without graphic output, etc.) *vs.* 0.04 seconds to compute the "unrefined" solution up to the same time. This is a factor of 29 more expensive. But the unrefined solution was worthless.

Thus, to study the efficiency of our method, we need to compare the computing time taken by our method with the computing time taken to produce (approximately) the same error on a uniform (fine) mesh. As a by-product, we will also be able to compare the memory taken in the two approaches.

Our method is simple. Instead of comparing a "refined" solution Σ with a solution computed on only its coarsest mesh, we compare Σ with a solution computed on a uniform fine mesh whose spacing is the same as the spacing of the Σ's finest mesh. If these two produce approximately the same error, then we have a valid comparison.

Because this is probably the most important result in this paper, we made this study on all three of our model problems. The result is approximately the same for all.

Table 1 shows the results. The errors are at $t = 3.6$ for P1 and P2, and at $t = 0.15$ for P3. For P1 and P2, this is the time just before the pulse leaves the spatial interval. (We will measure the error at these times in the rest of this section.) The local error tolerance was 0.001 for P1 and P2 and 0.01 for P3. We used Richardson extrapolation. The maximum error is the maximum over all components of the (global) error n-vector, and over all grid points existing at a given time. The $l_2(x)$ norm of v was given in Section 4. Here we alter the definition by taking the l_2 norm of each component of the solution, and then taking the maximum of the n results. The $l_2(x)$ norm of the error is analogous. We used upwind/downwind boundary conditions for P1 and P2 (see Section 8.1).

In our tables, a maximum mesh level of 1 signifies that only the coarse mesh is present (no refinement), 2 signifies one additional level of refinement, and so forth. The times reported are CPU seconds on a CDC 7600. (Since this machine runs only in batch, these times are highly reproducible between runs, and we did not need to take average times.) The storage used is for solution values only, and is the maximum storage used at the (refinement) level listed, per solution component, for all times. Since the coarse mesh is static, it always uses 81 or 101 locations. The total listed is the sum over all levels.

We see that in terms of computer time our method is 3 to 5.5 times as efficient as using a uniform fine mesh which produces the same error. In terms of memory, a factor of 1.7 to 2.2 is gained. At first it might seem surprising that our method could be more efficient, since it requires much greater overhead (to estimate the error and adjust refinements) than the uniform mesh method. This is compensated for, however, by being able to take large time steps in unrefined regions. Most of the computing time is spent in advancing the solution on the finest refinements. The error estimation and mesh manipulations are almost free. That is why it is so important to refine only where necessary.

In general, of course, the specific efficiency factor depends primarily on the fraction of the region needing refinement, and other factors such as the local error tolerance, when (for which t) we are doing the comparison, the wave speed, and so forth.

These results show why we need to create, delete, merge, and separate refinements. If we did not allow merging and separating, then in P2, we would have had to refine the whole region when the pulses were entering or leaving the region and this would degrade the efficiency. Similarly, in P3, we would have had to use a single refinement encompassing all the non-constant features of the solution.

Our mesh refinement algorithm reduces the (maximum) number of mesh points needed to achieve a given accuracy, and this naturally reduces the amount of work, but does the amount of effort *per mesh point* decrease? Table 1 also gives this figure, obtained by dividing the computer time by the maximum number of mesh points used. It is clear that in all cases the work per mesh point is decreased by a factor of approximately two (the notation $n-m$ means $n \cdot 10^m$).

It might be argued that we obtained our results only by adjusting or tuning the parameters N, M. To refute this charge, we have shown several different values of N and M. This shows that although we cannot easily determine the optimal N, M, even suboptimal choices still yield a significant savings in execution time.

8.5 Behavior as $h \to 0$. For our mesh refinement algorithm we can study two types of convergence. In all cases we let $\lambda = k_l / h_l$ = constant, independent of l. Thus N and M are fixed. We shall assume the exact solution is sufficiently smooth.

(a) We can keep the local error tolerance δ and the maximum refinement level Λ fixed, and let the largest spatial step $h = h_1$ approach zero. If we take a sufficiently large value for Λ, then the algorithm will refine as much as it needs to. Furthermore, (for smooth solutions) our method then has a property which leads to simplified analysis: *For sufficiently small h_1, our refined mesh becomes a uniform mesh.* This type of convergence is not desirable, since the advantages of refinement are ultimately lost.

(b) In the second method we let $h_1 \to 0$ and choose δ as a function of h_1, so that $\delta \to 0$ also. (Alternatively, we could let $\delta \to 0$ and choose h_1 as a function of δ.) If one knows the order of the global error, one could choose $\delta = C(h_1)^p$ for some constant C. This is certainly the theoretically most appealing method. If we use this method, then the grid does *not* approach a uniform coarse grid as $h_1 \to 0$. Rather, the ratio of the width of any refinement to the width of its parent should approach a constant as $h_1 \to 0$. However, for checking the asymptotic behavior of the program we shall first use method (a), since it does not beg the question by assuming the behavior of the global error.

We first keep the maximum number of refinement levels constant, and less than necessary (for the method to refine as much as possible), and study the first type of convergence. Table 2 shows these results on P1 using $\lambda_1 = 0.8$, $N = M = 4$, three-level Richardson extrapolation, and local error tolerance = 0.001. For the smaller values of h, the $O(h^2)$ behavior of the errors (both maximum and l_2) is apparent.

Next we do the same test, but choose the maximum number of levels large enough so that the method refines as much as possible. The maximum level is 5 here. The convergence is $O(h^2)$ for the maximum error with the coarse mesh size going from $N_0 = 80$ to 160. But the grid is approaching a uniform mesh as $h \to 0$ and this slows down the convergence. As $h \to 0$ the number of levels used approaches 1.

Using method (b), we let $h \to 0$ and let $\delta = 0(h^2)$. Here we see the convergence is faster, and the maximum error is finally $O(h^2)$. The l_2 error does not behave as well.

8.6 Estimating the Local Truncation Error in the Interior. In [5] we compared the Richardson method and differences for estimating the error in the interior of refinements on problem P1. We found that there was very little difference in efficiency between these methods. The use of differences was slightly more efficient. But the greater convenience of Richardson for interior approximations far outweighs any small efficiency differences.

8.7 Estimating the Local Truncation Error at Boundaries. In this section we will vary our boundary approximation and our method of error estimation at the boundary, while using the Richardson method in the interior on problem P2.

We will use upwind/downwind differencing and first-order extrapolation. For the former we will estimate the error by using both the modified Richardson method (Section 3.3) and differences. For extrapolation we can only use differences. The results are shown in Table 3. In all cases, the number of intervals on the coarsest mesh is 80, the maximum number of refinement levels is 5, and the refinement ratios $N = M = 4$. In all cases the fifth refinement level was not used. U/D signifies upwind/downwind differencing, and Rich. signifies the modified Richardson method. The memory occupied by solution values is the maximum total over all refinement levels for one component of the solution. All other parameters not shown are the same as in the computation for problem P2 in Table 1.

Clearly, the different boundary approximations and error estimation methods produce approximately the same results. This supports our claim that our method of adaptively handling boundaries is quite general.

8.8 How Often Should the Local Truncation Error Be Checked?. In Section 2 we used subsequences to describe the times at which we estimate the local truncation error (and possibly alter refinements). In this section we shall show that for Problem P1 it is unwise to monitor the local truncation error more often than every coarse time step.

Table 4 shows the results of these computations for Problem P1. All parameters not mentioned are the same as in the computations for Table 1. The meaning of (a) under "tolerance frequency" in Table 4 is how many coarse time steps occur between checks of the local error. The column (b) has two different meanings, depending on column (a). If column (a) is 1 then we check the truncation error at any time a refinement whose level is less than or equal to (b) is about to be advanced. Thus, in these cases we check *more often* then every coarse time step. Table 4 shows that this is very costly and produces no benefits whatever.

If (a) in Table 4 is greater than one, a one in column (b) signifies that we check all refinements every (a) coarse time steps. If column (a) is greater than one and (b) is greater than one, we check refinements with levels greater than or equal to (b) every coarse time step, and all others every (a) coarse time steps. Of course, in all cases in this table, the buffers mentioned in Section 2.6 have to be modified, in a way analogous to the argument given there.

Our results for these cases show very little difference from checking every coarse time step, until the checking frequency becomes too seldom (as in case (a) = 6, (b) = 1). Then the accuracy starts to deteriorate, because a pulse may enter the boundary before it is enclosed in refinement(s). (The algorithm could easily be modified to check the boundaries at every coarse time step, but we did not do this.)

We conclude that for this problem we may as well check the local error every coarse time step, although this may depend on factors such as the spacing of the coarse mesh, the wave speed, and the presence of forcing functions (terms kF in (2.1)). Also, the results may be radically different in more than one space dimension [2].

8.9 Linear vs. Quadratic Interpolation. An implementation detail we considered is whether to use linear or quadratic interpolation when a level l refinement moves into a region formerly occupied only by a level $l-1$ refinement. In [5] we found that there is practically no difference. We used quadratic interpolation elsewhere in this section.

Acknowledgments. I am grateful to my research advisor, Professor Joseph Oliger, for suggesting the topic of this thesis, and for his encouragement and advice during its preparation. I also wish to thank Gene Golub, John Herriot, Joseph Steger, George Homsy, and Paul Concus for encouragement and support. Also my thanks to Marsha Berger, Tony Chan, Phil Colella, and Gerald Hedstrom for suggestions and technical discussions.

I acknowledge computing facilities used: at the Stanford Linear Accelerator Center, operated for the U. S. Department of Energy by Stanford University; and at the Computer Center of the Lawrence Berkeley Laboratory, operated for the Department of Energy by the University of California.

Finally, I gratefully acknowledge financial support provided by Stanford University as a teaching assistant, by the Office of Naval Research under contract N00014-75-C-1132, and by the Director, Office of Energy Research, Office of Basic Energy Sciences, Engineering, Mathematical and Geosciences Division of the U.S. Department of Energy under contract DE-AC03-76SF00098.

REFERENCES

[1] I. Babushka, and W. C. Rheinboldt, *Error Estimates for Adaptive Finite Element Computations*, SIAM J. Numer. Anal., 15 (1978), pp. 736-754.

[2] M. Berger, *Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations*, Ph.D. thesis, Stanford University Computer Science Report STAN-CS-82-924, August, 1982.

[3] M. Berger, W. Gropp, and J. Oliger, *Mesh Generation for Time-Dependent Problems: Criteria and Methods*, Proc. Workshop on Numerical Grid Generation Techniques for Partial Differential Equations, NASA Conference Publication 2166, October, 1981.

[4] M. Berger, W. Gropp, and J. Oliger, *Stability Analysis*, to appear.

[5] J. Bolstad, *An Adaptive Finite Difference Method for Hyperbolic Systems in One Space Dimension*, Ph.D. thesis, Lawrence Berkeley Laboratory LBL-13287 and Stanford University Computer Science Department STAN-CS-82-899, June, 1982.

[6] J. U. Brackbill and J. Saltzman, *Adaptive Zoning for Singular Problems in Two Dimensions*, J. Comp. Phys., 46 (1982), pp. 342-368.

[7] G. Browning, H. O. Kreiss, and J. Oliger, *Mesh Refinement*, Math. Comp., 27 (1973), pp. 29-39.

[8] P. Budnik and J. Oliger, *Algorithms and Architecture*, High Speed Computer and Algorithm Organization, D. J. Kuck, D. H. Lawrie, and A. H. Sameh, eds., Academic Press, New York, 1977, pp. 355-370.

[9] M. Ciment, *Stable Difference Schemes with Uneven Mesh Spacings*, Math. Comp., 25 (1971), pp. 219-227.

[10] A. J. Cook and L. J. Shustek, *A User's Guide to MORTRAN2*, Computation Research Group, Stanford Linear Accelerator Center, Stanford, CA, 1975.

[11] S. Davis and J. Flaherty, *An Adaptive Finite Element Method for Initial Boundary-Value Problems for Partial Differential Equations*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 6-27.

[12] C. de Boor, *Good Approximation by Splines with Variable Knots. II*, Conference on the Numerical Solution of Differential Equations, G. Watson, ed., Lecture Notes in Mathematics 363, Springer Verlag, New York, 1974, pp. 12-20.

[13] H. A. Dwyer, R. J. Kee, and B. R. Sanders, *Adaptive Grid Method for Problems in Fluid Mechanics and Heat Transfer*, AIAA J., 18 (1980), pp. 1205-1212.

[14] B. Engquist and T. Smedsaas, *Automatic Computer Code Generation for Hyperbolic and Parabolic Differential Equations*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 249-259.

[15] S. Feldman, *The Programming Language EFL*, Bell Laboratories Comp. Sci. Tech. Rep. No. 78, 1979.

[16] D. Gannon, *Self Adaptive Methods for Parabolic Partial Differential Equations*, Dept. of Computer Science, Univ. of Illinois, 1980.

[17] R. J. Gelinas, S. K. Doss, and K. Miller, *The Moving Finite Element Method: Applications to General Partial Differential Equations with Multiple Large Gradients*, J. Comp. Phys., 40 (1981), pp. 202-249.

[18] W. D. Gropp, *A Test of Moving Mesh Refinement for 2-D Hyperbolic Problems*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 191-197.

[19] W. D. Gropp, *Preprocessor Language*, to appear.

[20] E. Grosse, *Software Restyling in Graphics and Programming Languages*, Stanford Univ. Comp. Sci. Report STAN-CS-78-663, 1978.

[21] B. Gustafsson, *The Convergence Rate for Difference Approximations to Mixed Initial Value Problems*, Math. Comp., 29 (1975), pp. 396-406.

[22] B. Gustafsson, H. O. Kreiss, and A. Sundström, *Stability Theory of Difference Approximations for Mixed Initial Boundary Value Problems, II*, Math. Comp., 26 (1972), pp. 649-686.

[23] G. W. Hedstrom and G. H. Rodrigue, *Adaptive-Grid Methods for Time-Dependent Partial Differential Equations*, Proceedings of the Conference on Multigrid Methods, Cologne, Lecture Notes in Computer Science, Springer Verlag, New York, 1981.

[24] S. S. Hu and W. E. Schiesser, *An Adaptive Grid Method in the Numerical Method of Lines*, Advances in Computer Methods for Partial Differential Equations IV, R. Vichnevetsky and R. S. Stepleman, eds., IMACS, New Brunswick, NJ, 1981, pp. 305-311.

[25] K. Jensen and N. Wirth, *Pascal User Manual and Report*, 2nd ed., Springer Verlag, New York, 1974.

[26] B. Kernighan, *RATFOR - a Preprocessor for a Rational Fortran*, Software -- Practice and Experience, 5 (1975), pp. 395-406.

[27] B. Kernighan, *Why Pascal is Not My Favorite Programming Language*, Computing Science Technical Report No. 100, Bell Laboratories, 1981.

[28] D. E. Knuth, *The Art of Computer Programming*, vol. 1, 2nd ed., Addison-Wesley, Reading, MA, 1973.

[29] H. O. Kreiss and J. Oliger, *Comparison of Accurate Methods for the Integration of Hyperbolic Equations*, Tellus, XXIV (1972), pp. 199-215.

[30] D. C. Lam and R. B. Simpson, *Modelling Coastal Effluent Transport Using a Variable Finite Difference Grid*, Advances in Computer Methods for Partial Differential Equations II, R. Vichnevetsky, ed., IMACS, New Brunswick, NJ, 1977, pp. 294-300.

[31] A. Majda and S. Osher, *Propagation of Error into Regions of Smoothness for Accurate Difference Approximations to Hyberbolic Equations*, Comm. Pure Appl. Math., 30 (1977), pp. 671-705.

[32] K. Miller and R. Miller, *Moving Finite Elements. I*, SIAM J. Numer. Anal., 18 (1981), pp. 1019-1032.

[33] J. Oliger, *Approximate Methods for Atmospheric and Oceanographic Circulation Problems*, Proc. Third International Symposium on Computing Methods in Applied Sciences and Engineering, R. Glowinski and J. Lions, ed., Lecture Notes in Physics 91, Springer Verlag, New York, 1979, pp. 171-184.

[34] J. Oliger, *Fourth Order Difference Methods for the Initial Boundary-Value Problem for Hyberbolic Equations*, Math. Comp., 28 (1974), pp. 15-25.

[35] J. Oliger, *Hybrid Difference Methods for the Initial Boundary-Value Problem for Hyperbolic Equations*, Math. Comp., 30 (1976), pp. 724-738.

[36] J. Oliger, *Constructing Stable Difference Methods on Piecewise Uniform Grids*, to appear.

[37] V. Pereyra and E. G. Sewell, *Mesh Selection for Discrete Solution of Boundary Problems in Ordinary Differential Equations*, Numer. Math., 23 (1975), pp. 261-268.

[38] B. Pierson and P. Kutler, *Optimal Nodal Point Distribution for Improved Accuracy in Computational Fluid Dynamics*, AIAA J., 18 (1980), pp. 49-54.

[39] M. M. Rai and D. A. Anderson, *Application of Adaptive Grids to Fluid-Flow Problems with Asymptotic Solutions*, AIAA J., 20 (1982), pp. 496-502.

[40] W. C. Rheinboldt and C. Mesztenyi, *On a Data Structure for Adaptive Finite Element Mesh Refinements*, ACM TOMS, 6 (1980), pp. 166-187.

[41] R. Richtmyer and K. W. Morton, *Difference Methods for Initial Value Problems*, 2nd. ed., Wiley, New York, 1967.

[42] L. M. Shampine and M. K. Gordon, *Computer Solution of Ordinary Differential Equations: the Initial Value Problem*, Freeman, San Francisco, 1975.

[43] G. Sod, *A Survey of Several Finite Difference Methods for Systems of Nonlinear Hyperbolic Conservation Laws*, J. Comp. Phys., 27 (1978), pp. 1-31.

[44] J. Steger and D. Chaussee, *Generation of Body-Fitted Coordinates Using Hyperbolic Partial Differential Equations*, SIAM J. Sci. Stat. Comput., 1 (1980), pp. 431-437.

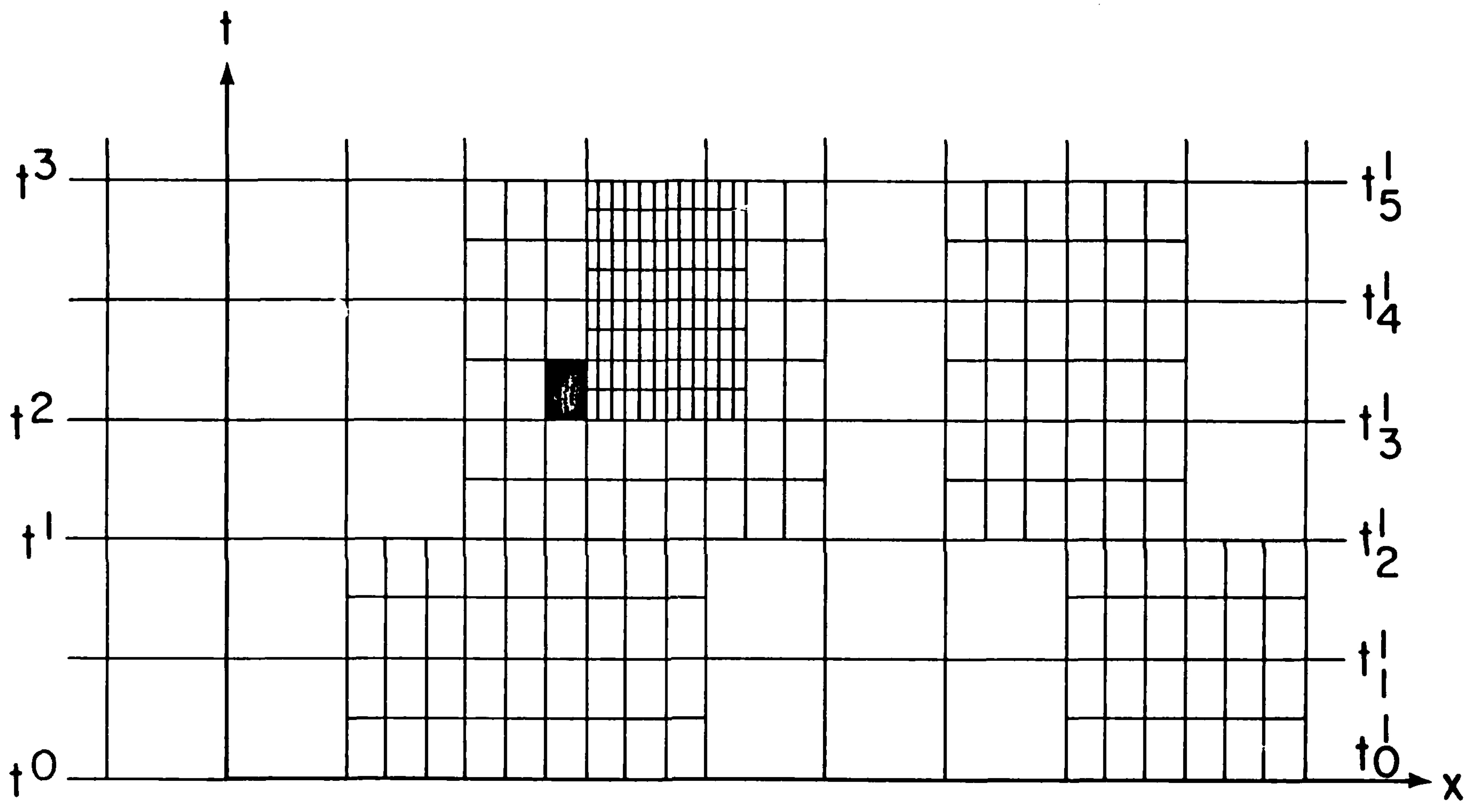
[45] J. Stein, *On the Usefulness of an Extensible Programming Language to Partial Differential Equations*, Advances in Computer Methods for Partial Differential Equations II, R. Vichnevetsky, ed., IMACS, New Brunswick, NJ, 1977, pp. 150-154.

[46] H. Stetter, *Global Error Estimation in Adams PC-Codes*, ACM TOMS, 5 (1979), pp. 415-430.

[47] A. B. White, *On the Numerical Solution of Initial/Boundary-Value Problems in One Space Dimension*, SIAM J. Numer Anal., 19 (1982), pp. 683-697.

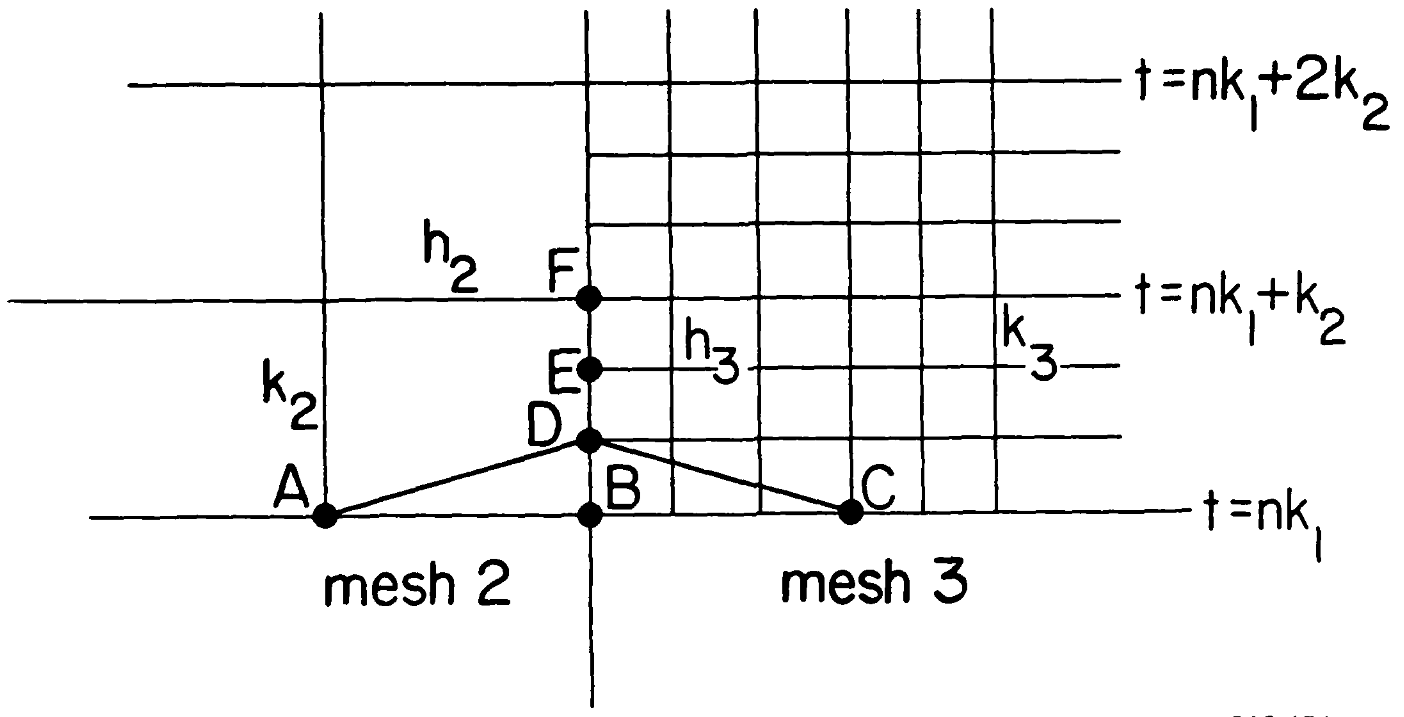
[48] P. Woodward and P. Colella, *The Numerical Simulation of Two-Dimensional Fluid Flow with Strong Shocks*, to appear in J. Comp. Phys.

[49] C. T. Zahn, *A User Manual for the MORTRAN2 Macro-Translator*, Computation Research Group, Stanford Linear Accelerator Center, Stanford, CA, 1975.



XBL 822-170

Figure 1 Mesh Structure



XBL 822-171

Figure 2 Coarse/Fine Interface

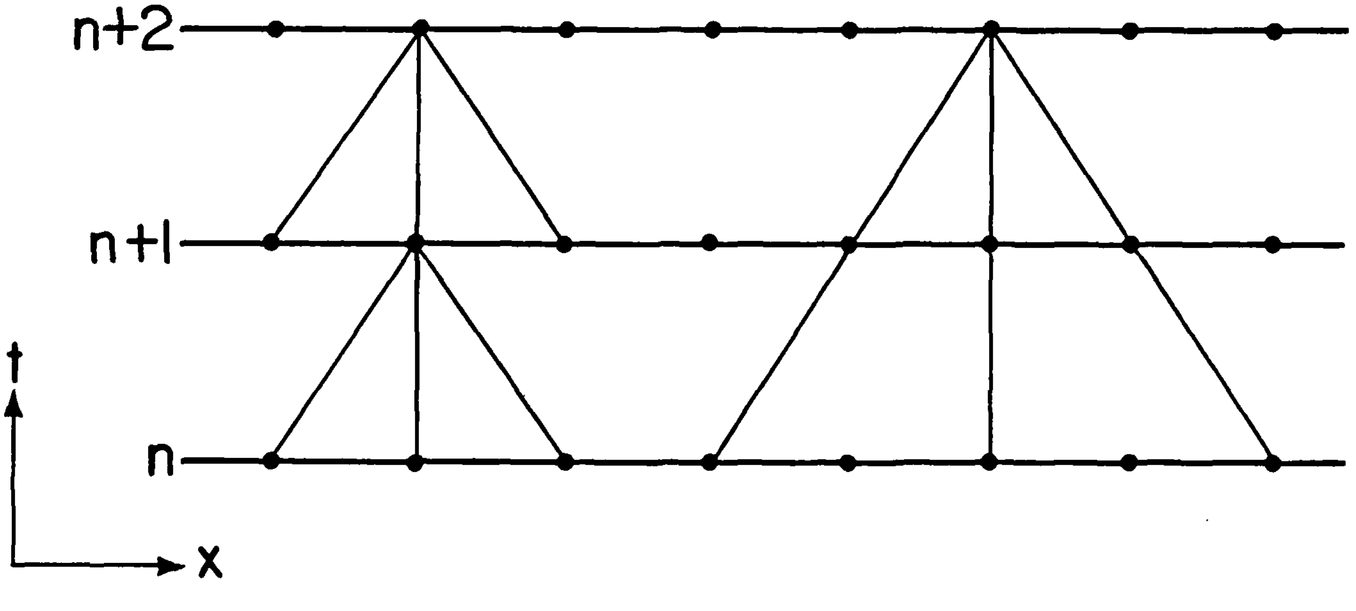


Figure 3 Richardson Method

XBL 822-165

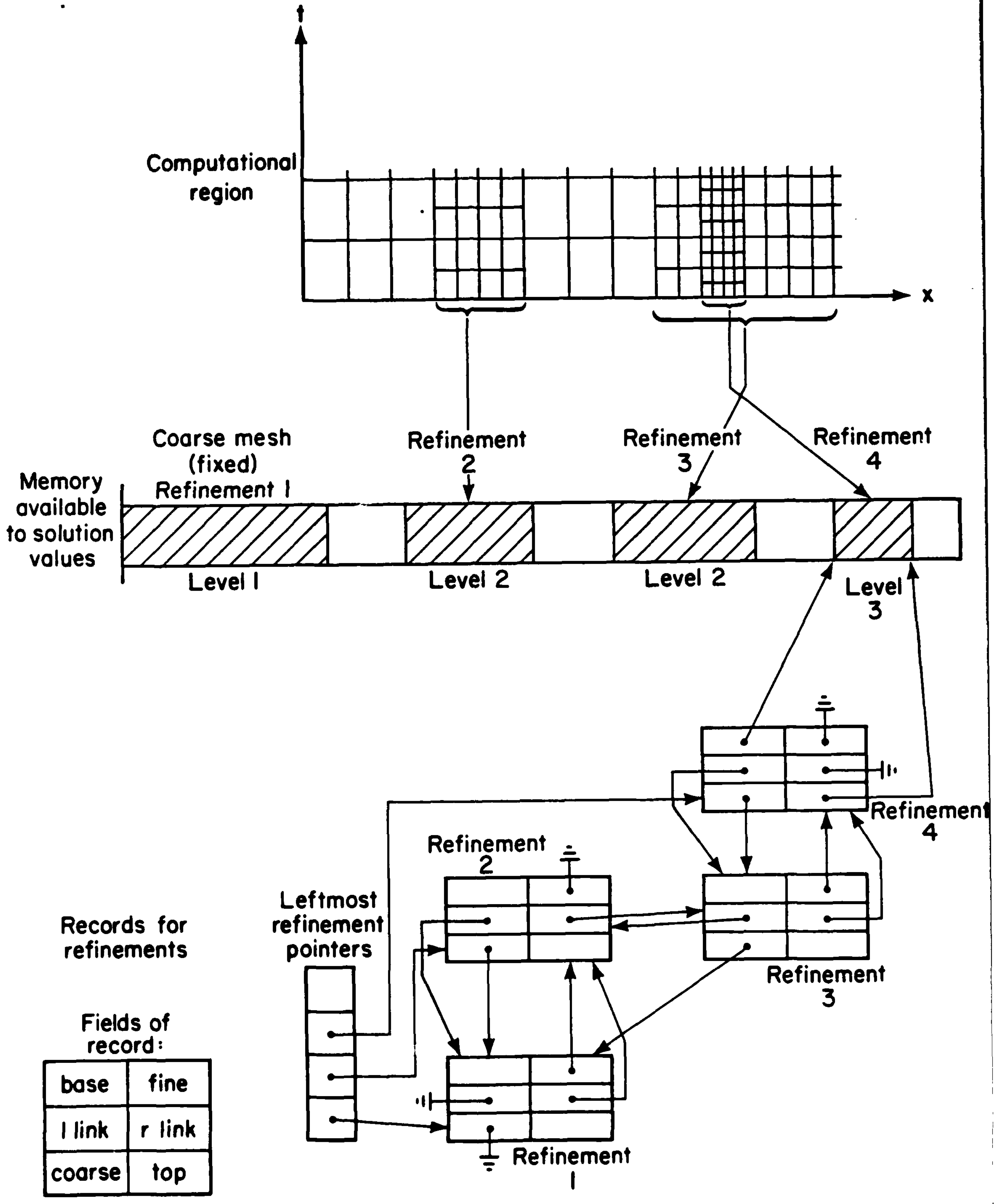


Figure 4 Data Structure: Tree and Vector of Deques

XBL 822-172

Problem No.	λ	No. Intervals on Coarsest Mesh	Maximum Refinement Level	N	M	ℓ_2 Error	Maximum Error	Time (sec.)	Memory Used	Work per Point
P1	.8	1280	1	-	-	5.89-3	2.01-2	7.67	1281	5.99-3
P1	.8	80	3	4	4	5.92-3	2.01-2	1.83	579	3.16-3
P1	.8	2000	1	-	-	2.42-3	8.20-3	18.4	2001	9.20-3
P1	.8	80	3	5	5	2.46-3	8.20-3	3.31	748	4.43-3
P2	.8	1280	1	-	-	8.33-3	2.01-2	13.0	1281	1.02-2
P2	.8	80	3	4	4	8.35-3	2.01-2	4.33	756	5.72-3
P2	.8	2000	1	-	-	3.43-3	8.22-3	30.0	2001	1.50-2
P2	.8	80	3	5	5	3.45-3	8.24-3	8.50	1009	8.42-3
P2	.8	2880	1	-	-	1.65-3	3.99-3	63.16	2881	2.19-2
P2	.8	80	3	6	6	1.67-3	4.00-3	15.19	1278	1.19-2
P3	.405405	1600	1	-	-	7.98-3	2.15-1	22.54	1601	1.41-2
P3	.405405	100	3	4	4	7.98-3	2.15-1	5.08	694	7.32-3
P3	.405405	2500	1	-	-	6.64-3	2.71-1	53.94	2501	2.16-2
P3	.405405	100	3	5	5	6.64-3	2.71-1	9.90	876	1.13-2

Table 1. Efficiency of the Method

No. of Intervals on Coarse Mesh	Maximum Refinement Level	N	M	Local Error Tol.	ℓ_2 Error	Maximum Error	ℓ_2-norm of Solution	Time (sec.)	Maximum Storage by Levels 1	2	3	4	Total
40	2	4	4	.001	.144	.391	.304	.178	41	161			202
80	2	4	4	.001	.0715	.230	.323	.647	81	321			402
160	2	4	4	.001	.0228	.0790	.328	.782	161	122			283
320	2	4	4	.001	.00592	.0201	.329	2.74	321	157			478
40	3	4	4	.001	.0228	.0790	.328	.592	41	161	158		360
80	3	4	4	.001	.00592	.0201	.329	1.89	81	321	181		583
160	3	4	4	.001	.00296	.00503	.329	3.77	161	118	281		560
320	3	4	4	.001	.000756	.00126	.329	13.5	321	161	445		927
40	5	4	4	.001	.00240	.00325		33.8	41	161	121	349	672
80	5	4	4	.001	.000743	.00126		12.35	81	321	177	497	1076
160	5	4	4	.001	.00260	.00330		22.5	161	101	277	433	972
320	5	4	4	.001	.000756	.00126		13.46	321	157	433	0	911
640	5	4	4	.001	.000454	.00140		28.8	641	257	374	0	1272
1280	5	4	4	.001	.000418	.00126		43.9	1281	433	0		1714
40	5	4	4	.01	.00388	.0109		2.78	41	161	109	213	524
80	5	4	4	.0025	.00139	.00498		8.20	81	321	169	305	876
160	5	4	4	.000625	.000438	.00139		26.7	161	641	289	493	1584
320	5	4	4	.000156	.000134	.000354		93.6	321	1076	545	869	2811

Table 2 Behavior of Global Error as $h_1 \to 0$

Boundary Approx.	Error Estimation	Local Error Tolerance	l_2 Error	Maximum Error	Time (sec.)	Memory Used
U/D	Rich.	.01	1.20-2	2.18-2	3.84	630
U/D	diff.	.01	1.21-2	2.18-2	3.87	630
extrap.	diff.	.01	1.18-2	2.18-2	3.93	642
U/D	Rich.	.001	8.02-4	1.34-3	42.3	1774
U/D	diff.	.001	8.02-4	1.34-3	41.9	1774
extrap.	diff.	.001	8.00-4	1.34-3	42.6	1814

Table 3. Error Estimation at Boundaries

Maximum Refinement Level	N	M	Tol.	Tol. (a)	Freq. (b)	ℓ_2 Error	Maximum Error	Time (sec.)	Maximum Storage by Levels 2	3	4	Total
4	4	4	.01	1	1	.0115	.0201	1.17	65	149	0	295
				1	2	.0115	.0201	1.64	65	125	0	271
				1	3	.0115	.0201	3.10	65	125	0	271
				2	1	.0112	.0202	1.16	69	173	0	323
				2	2	.0119	.0201	1.15	69	149	0	299
				2	3	.0112	.0202	1.24	69	173	0	323
				3	1	.0119	.0238	1.26	81	205	0	367
				3	2	.0114	.0219	1.21	81	161	0	323
				3	3	.0119	.0238	1.38	81	205	0	367
				4	1	.0107	.0241	1.31	85	221	0	387
				5	1	.0129	.0380	1.41	89	249	0	419
				6	1	.0235	.0857	1.48	93	281	0	455
3	6	6	.001	1	1	.00120	.00398	5.62	481	355		917
				1	2	.00120	.00398	7.86	481	313		875
				1	3	.00120	.00398	7.90	481	313		875
				2	1	.00120	.00398	6.10	481	415		977
				2	2	.00120	.00398	5.68	481	355		917
				2	3	.00120	.00398	6.06	481	415		977
				3	1	.00121	.00398	6.52	481	475		1037

Table 4 How Often Should the Local Truncation Error Be Checked?

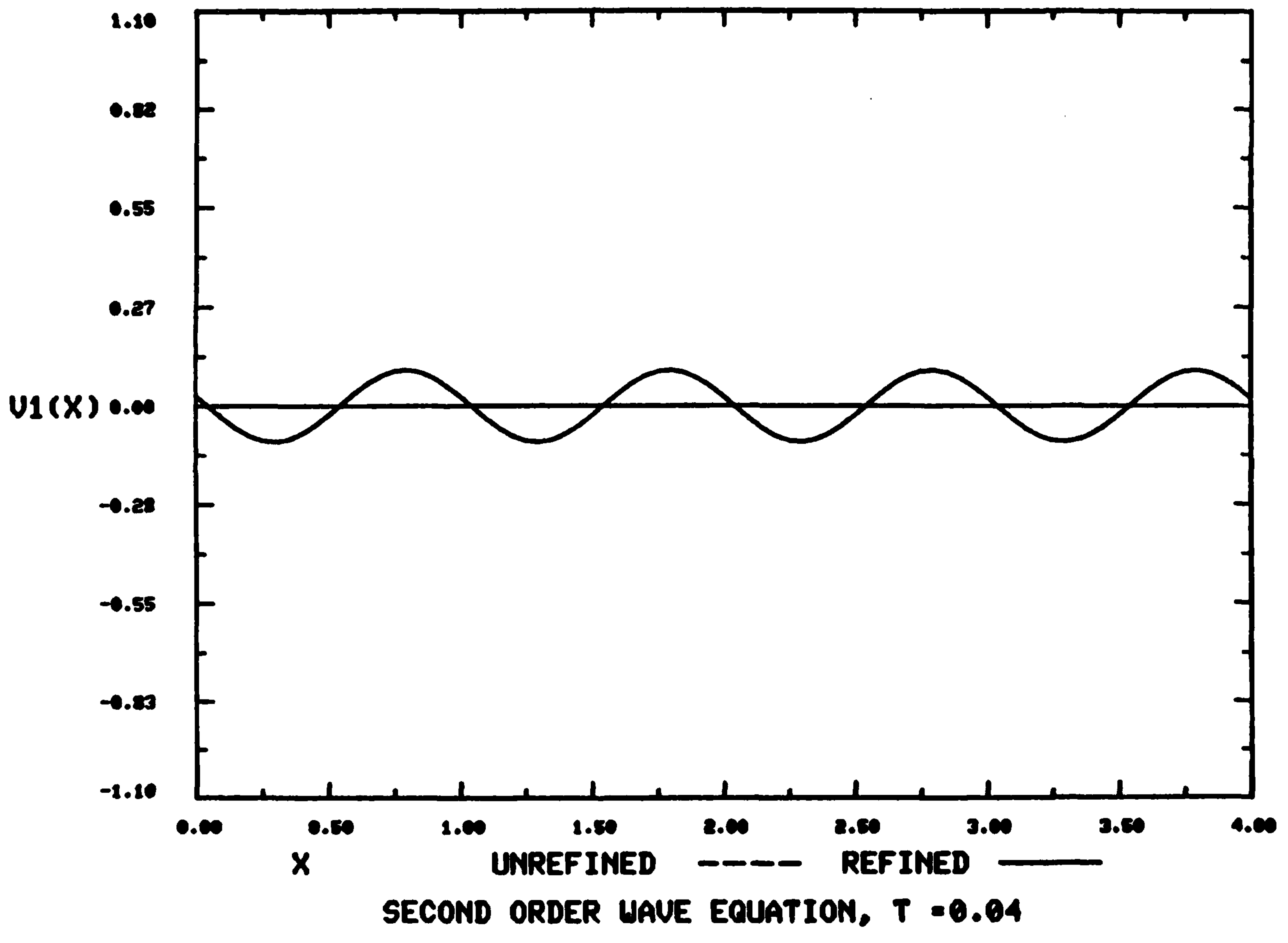
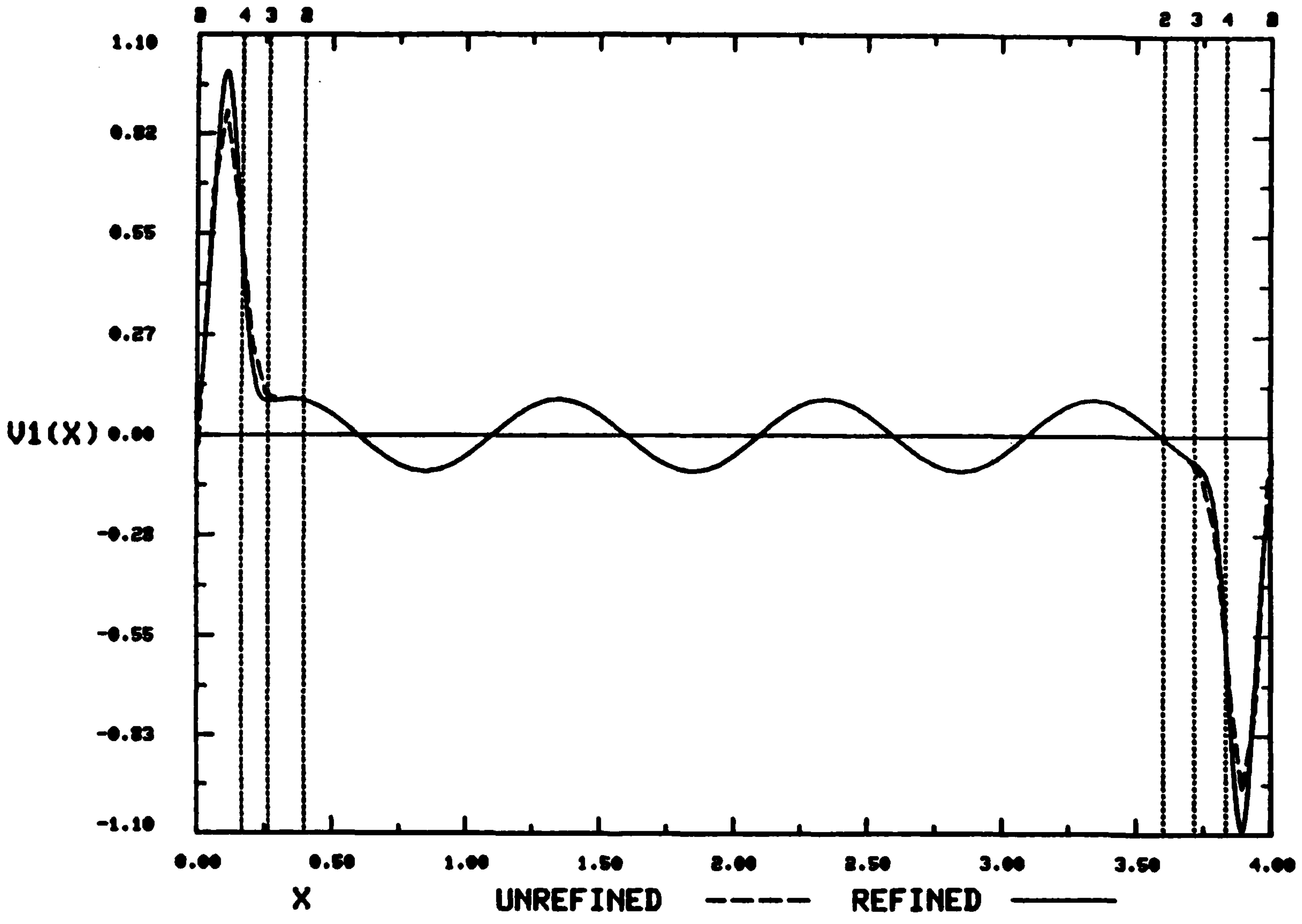
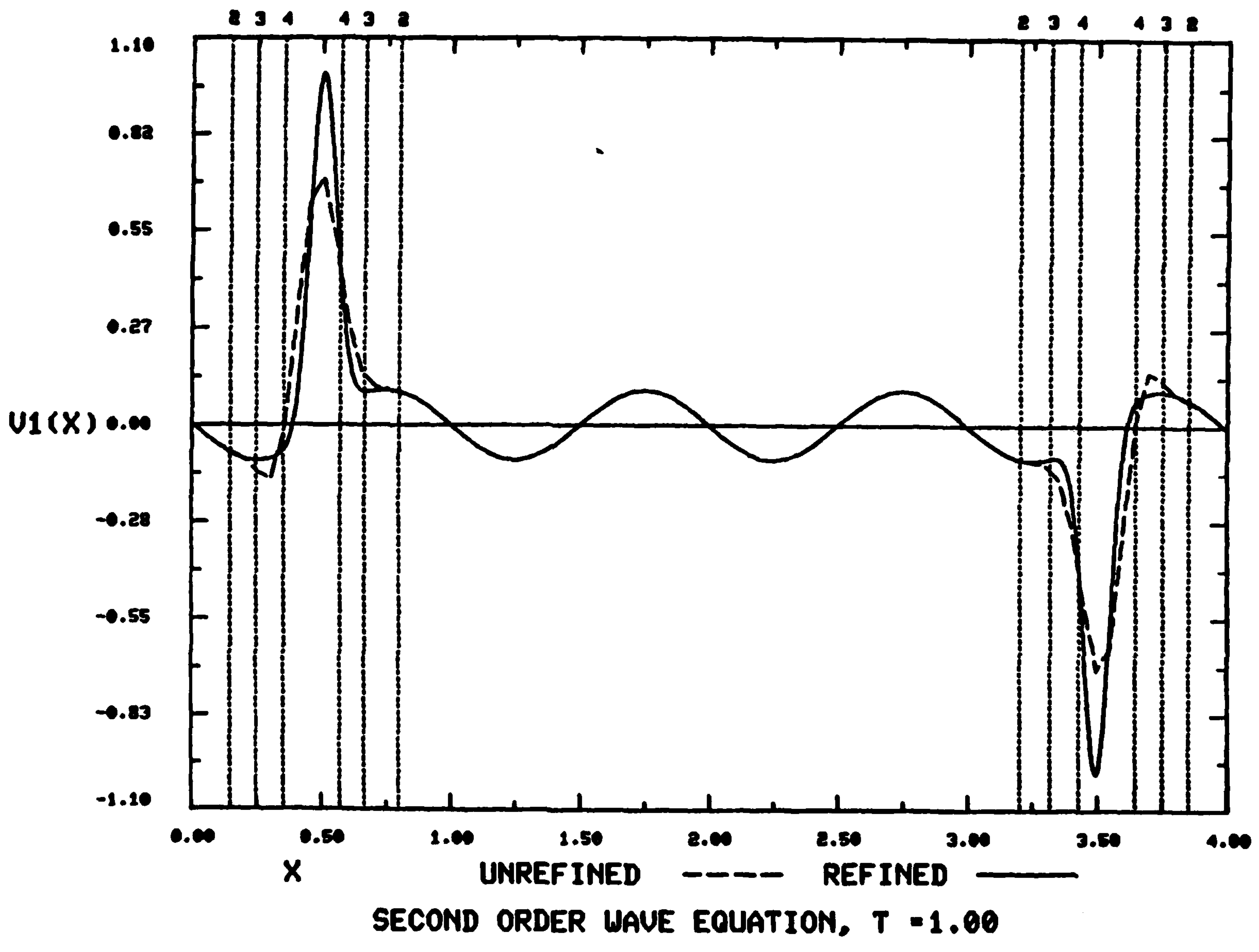

1.10
0.82
0.55
0.27
V1(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ---- REFINED ——
SECOND ORDER WAVE EQUATION, T =0.04

Figure 5(a)



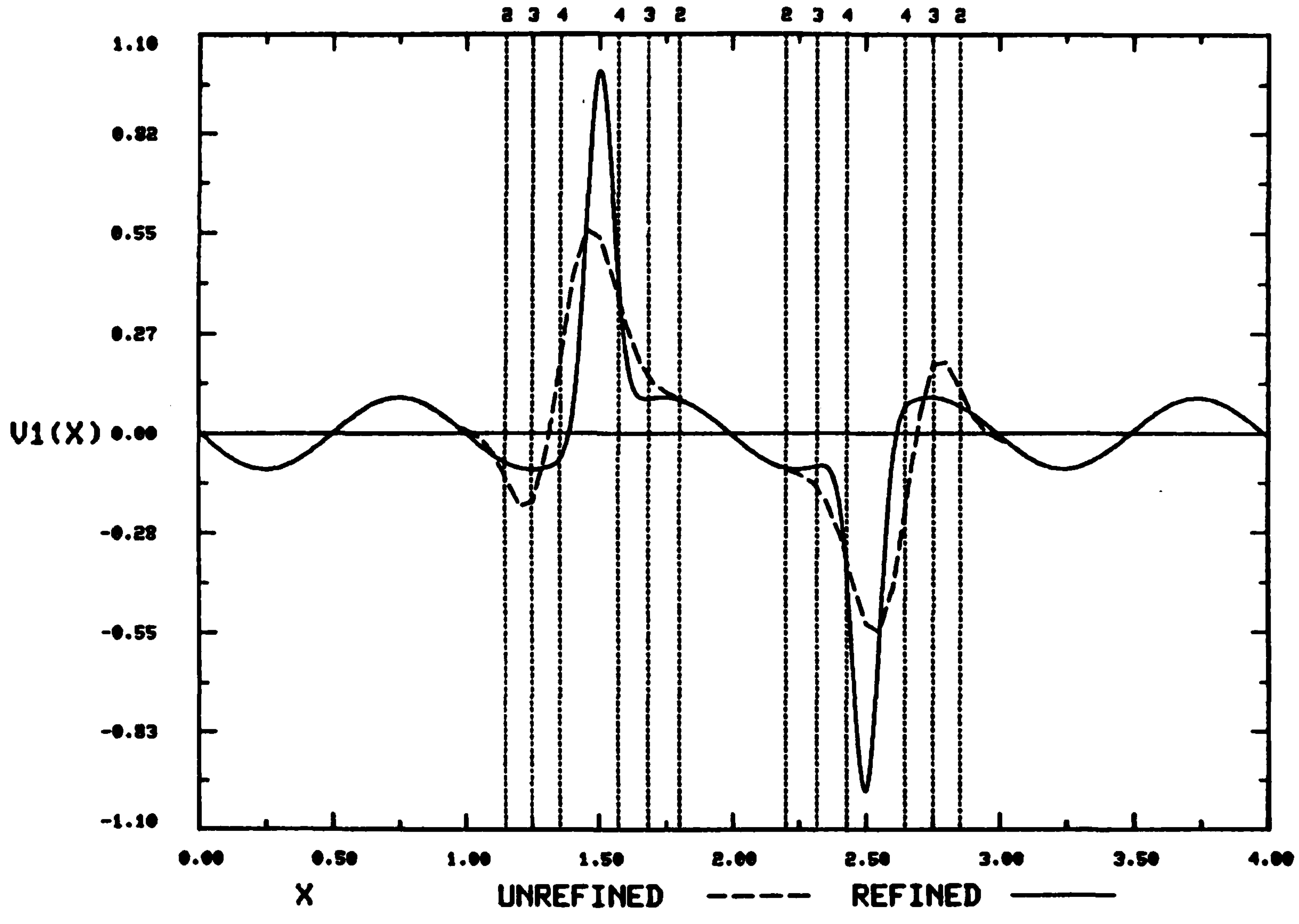
SECOND ORDER WAVE EQUATION, T =0.60

Figure 5(b)



SECOND ORDER WAVE EQUATION, T =1.00

Figure 5(c)



SECOND ORDER WAVE EQUATION, T =2.00

Figure 5(d)

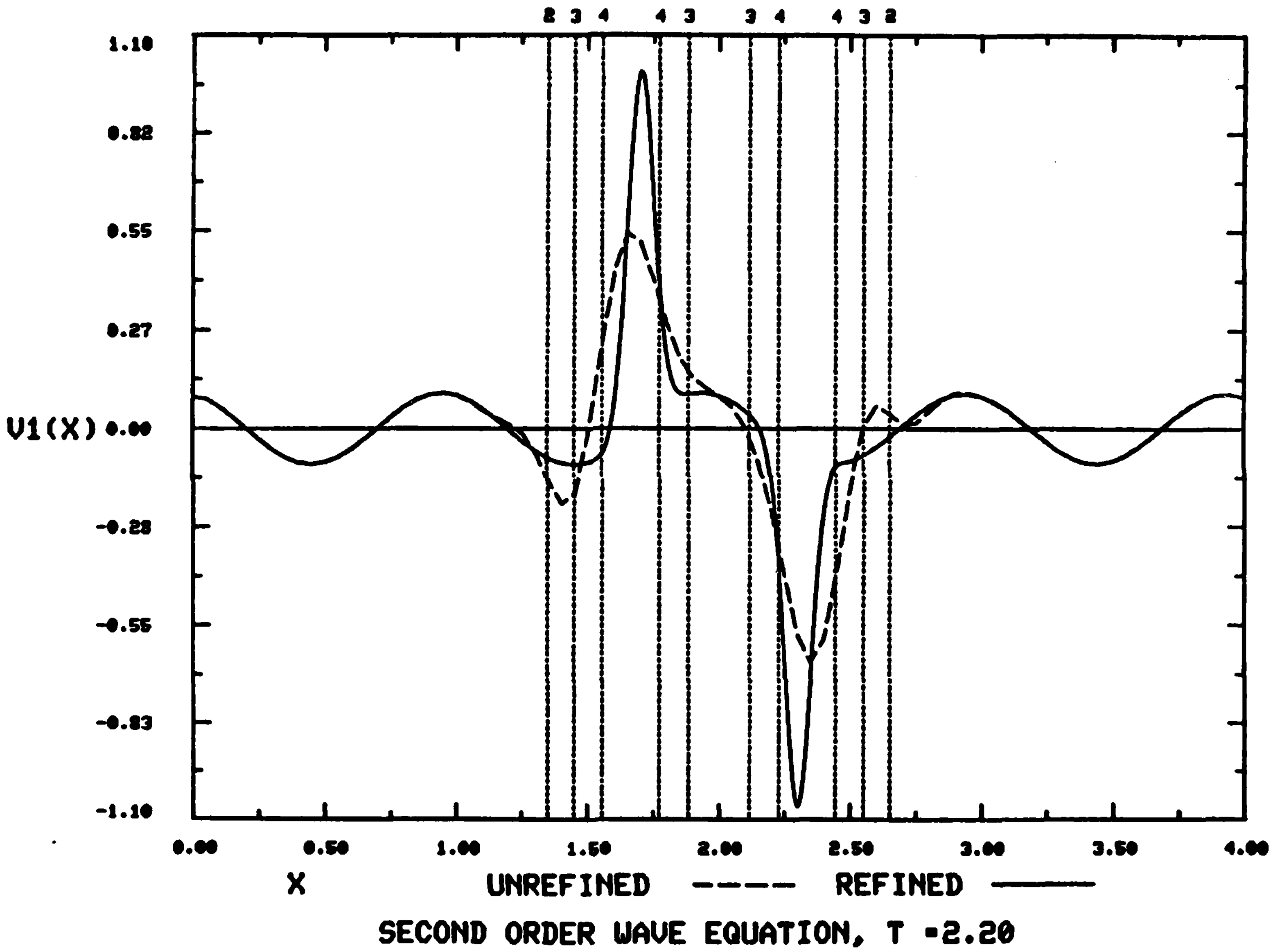
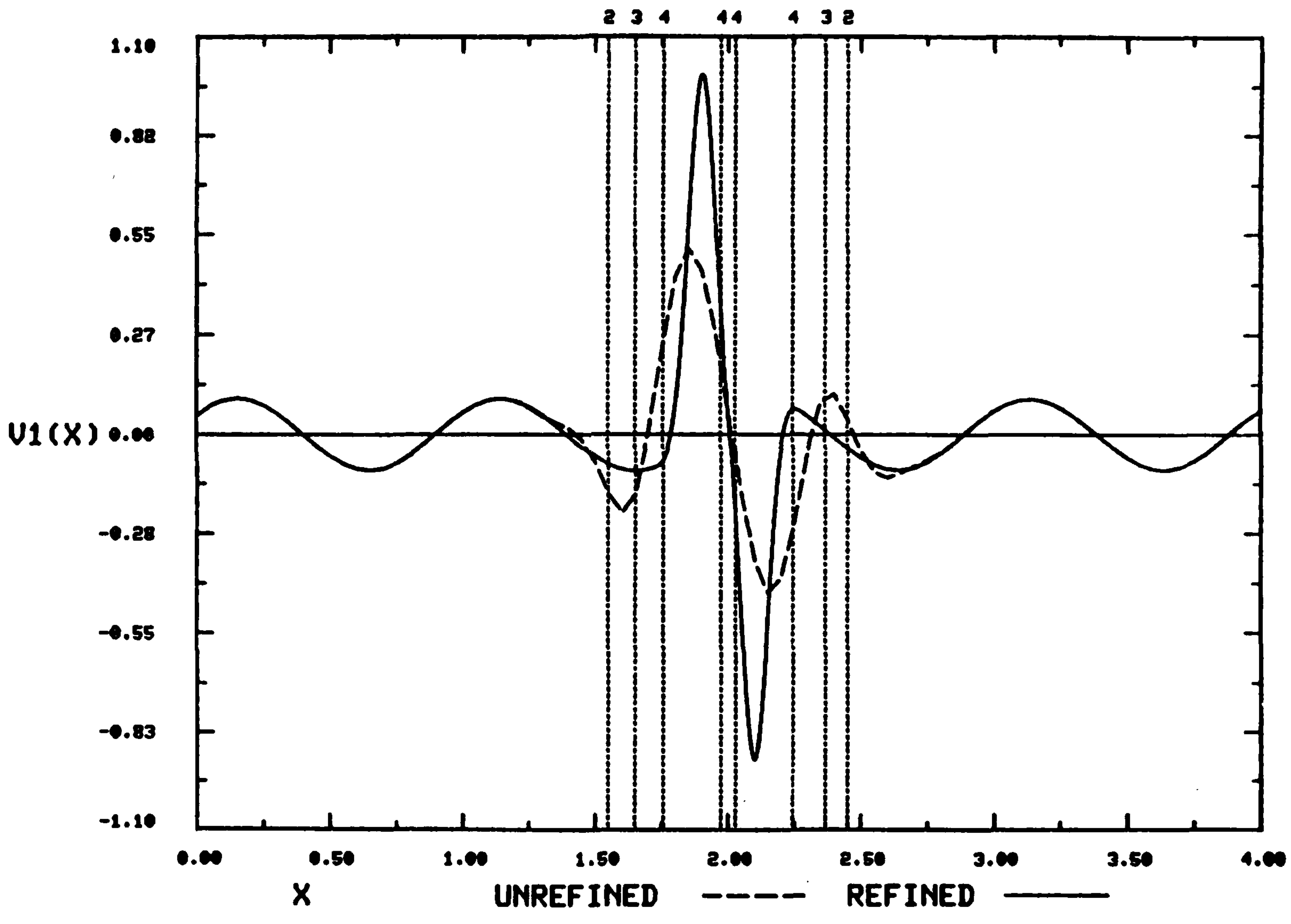

2 3 4 4 3 3 4 4 3 2
1.10
0.82
0.55
0.27
V1(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ---- REFINED ——
SECOND ORDER WAVE EQUATION, T =2.20

Figure 5(e)



SECOND ORDER WAVE EQUATION, T =2.40

Figure 5(f)

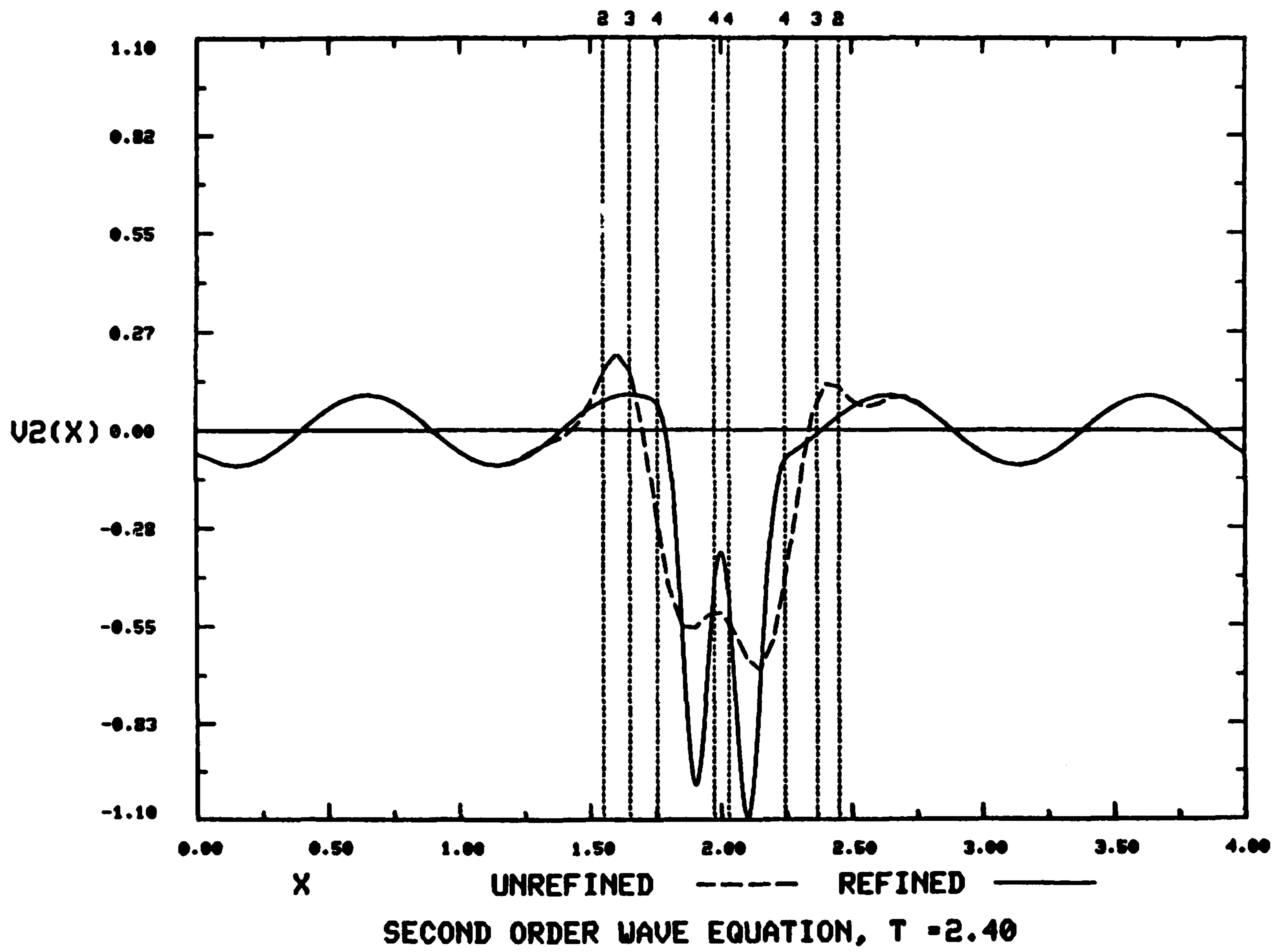

2 3 4 44 4 3 2
1.10
0.82
0.55
0.27
V2(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ----
REFINED ——
SECOND ORDER WAVE EQUATION, T =2.40

Figure 5(g)

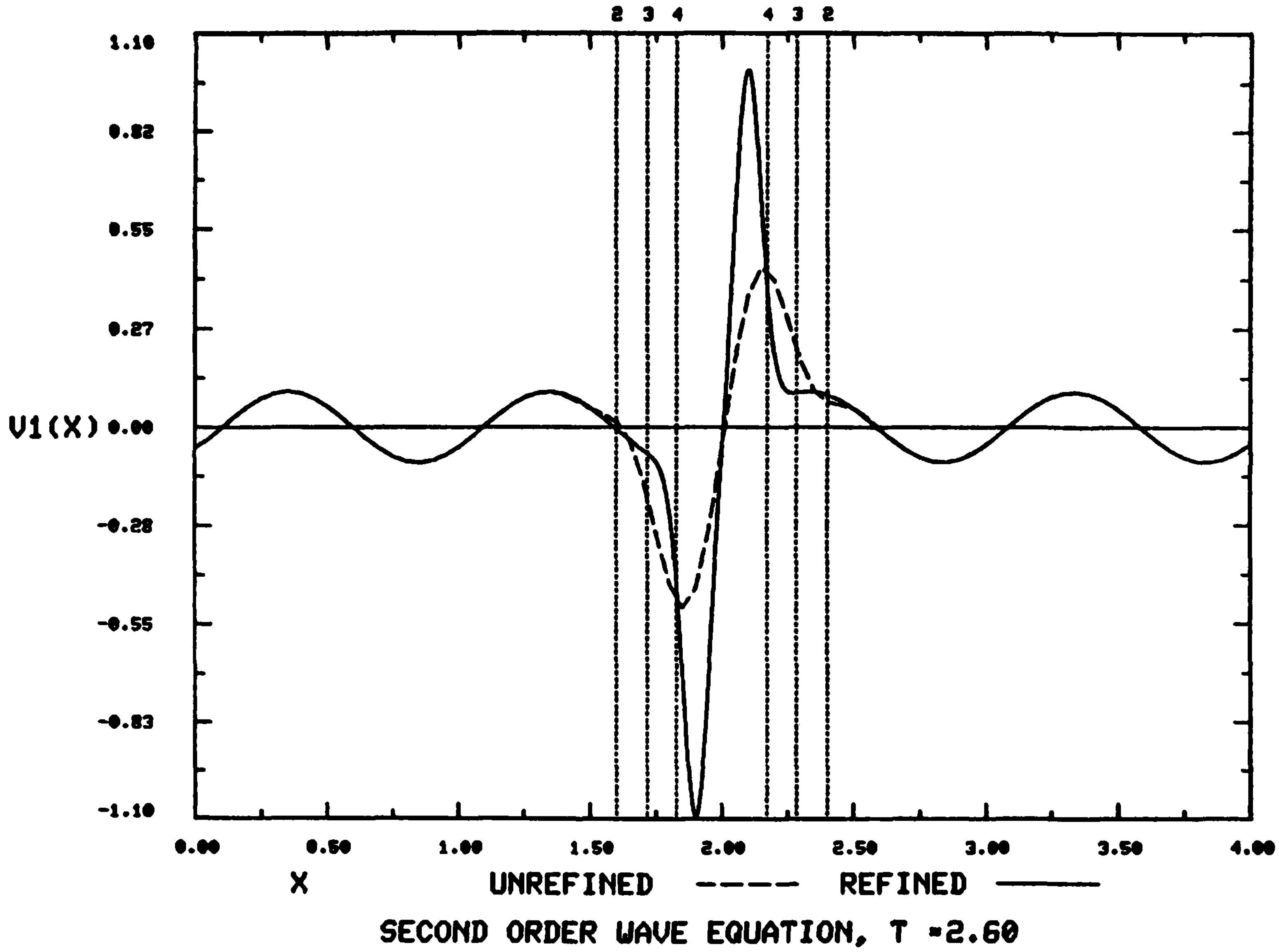

2 3 4 4 3 2
V1(X)
1.10
0.82
0.55
0.27
0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ---- REFINED ——
SECOND ORDER WAVE EQUATION, T =2.60

Figure 5 (h)

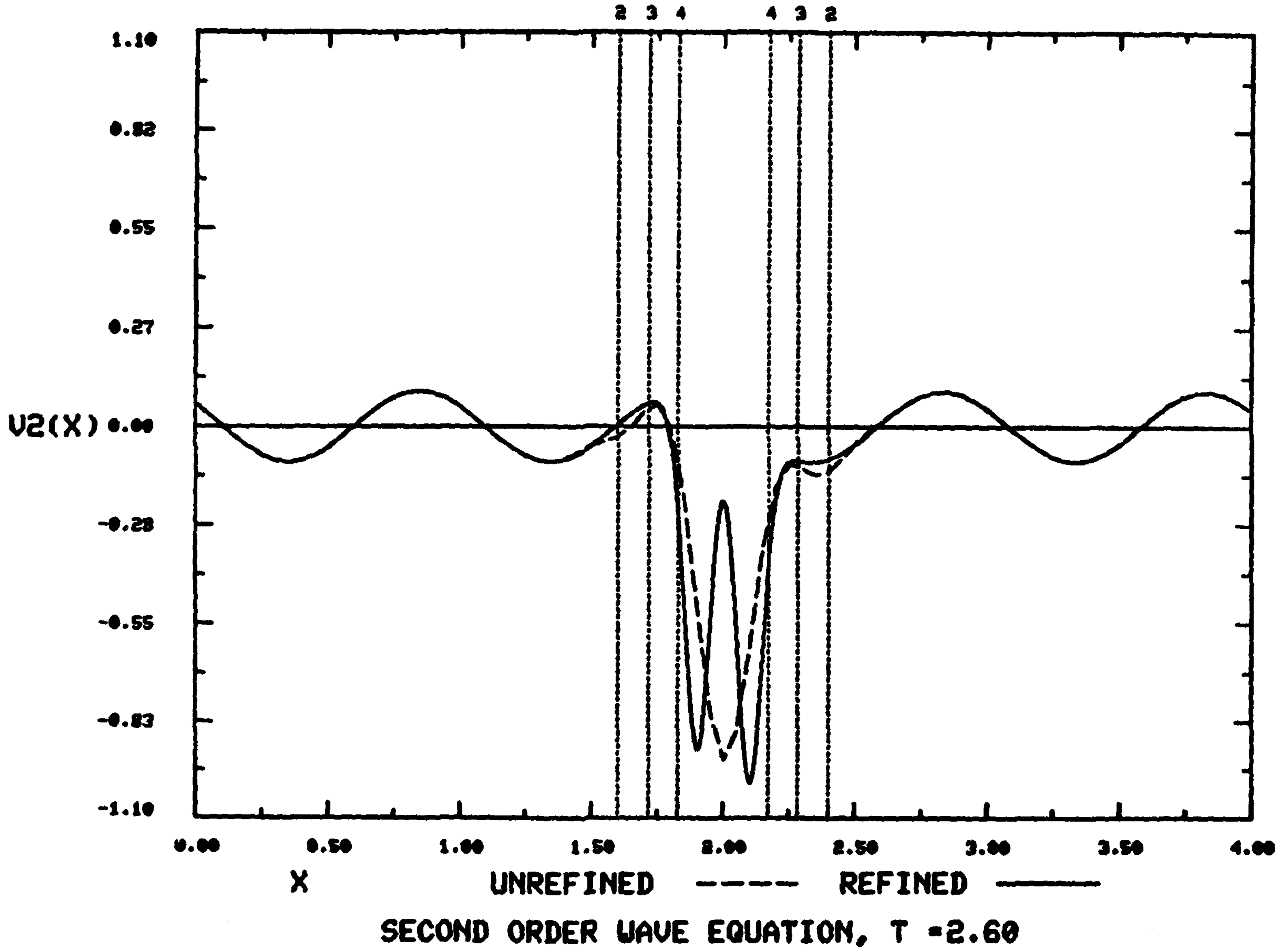
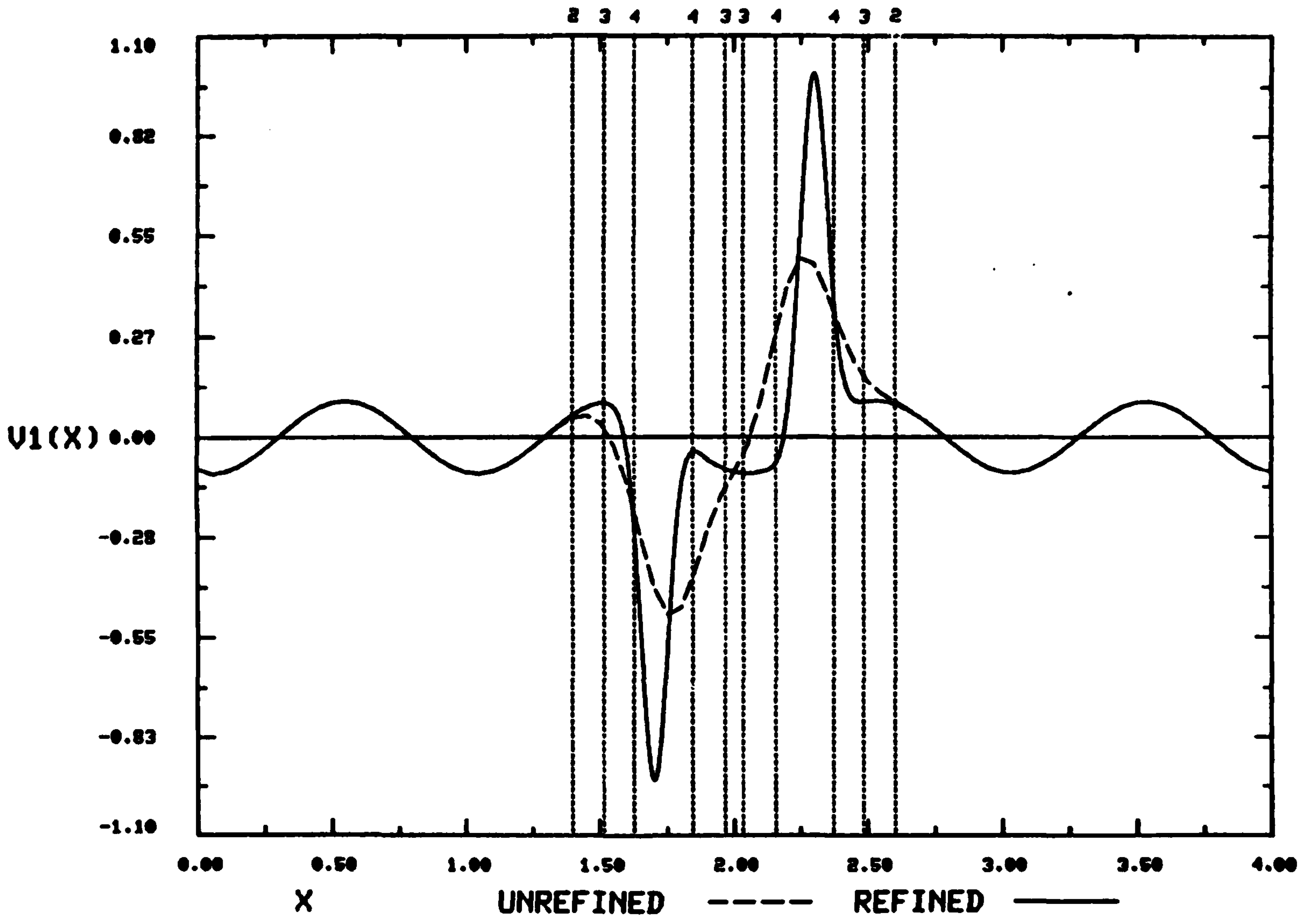

2 3 4 4 3 2
1.10
0.82
0.55
0.27
V2(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ----- REFINED ——
SECOND ORDER WAVE EQUATION, T =2.60

Figure 5(i)



SECOND ORDER WAVE EQUATION, T =2.80

Figure 5(j)

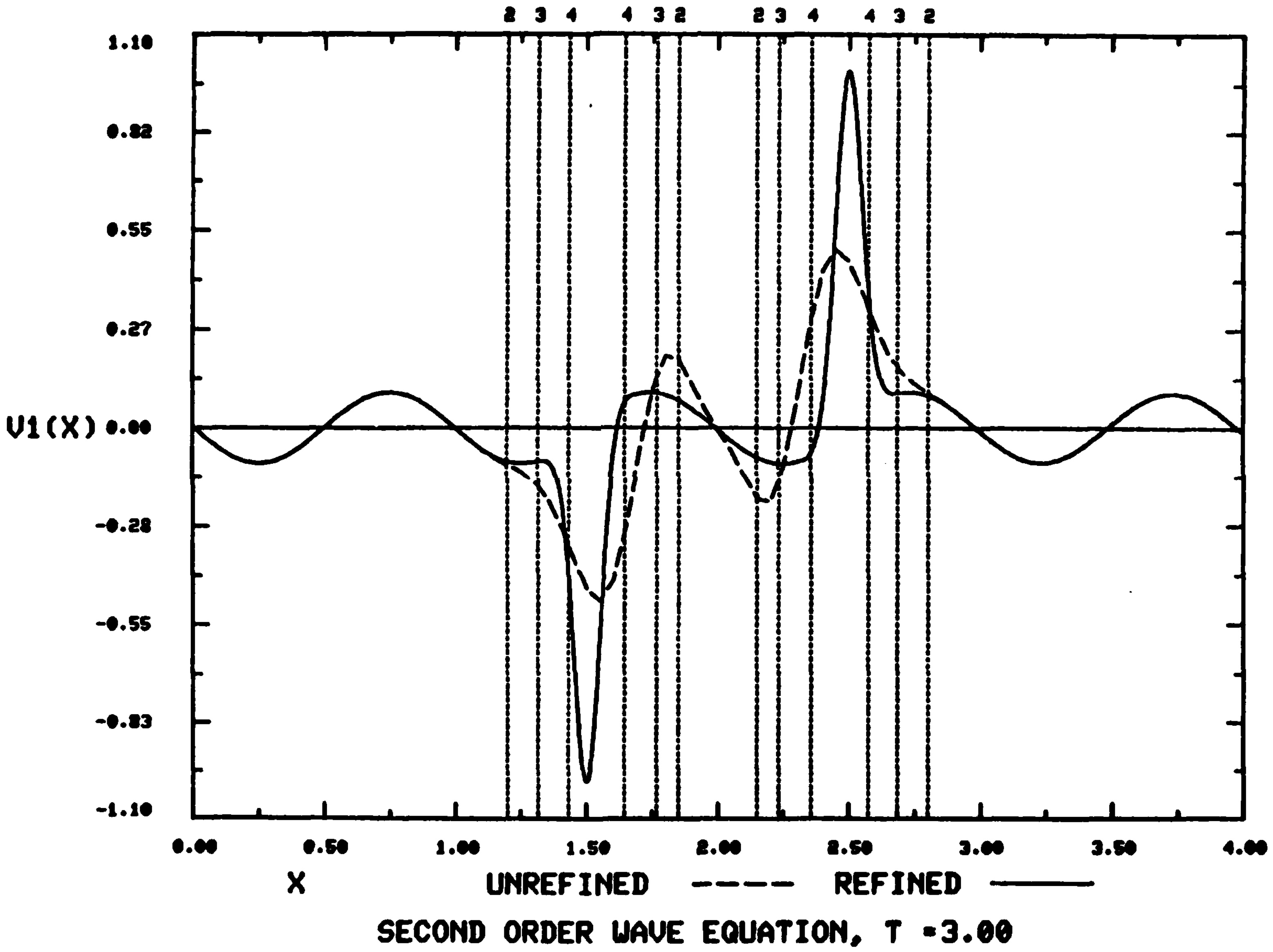

2 3 4 4 3 2 2 3 4 4 3 2
1.10
0.82
0.55
0.27
V1(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ---- REFINED ——
SECOND ORDER WAVE EQUATION, T =3.00

Figure 5(k)

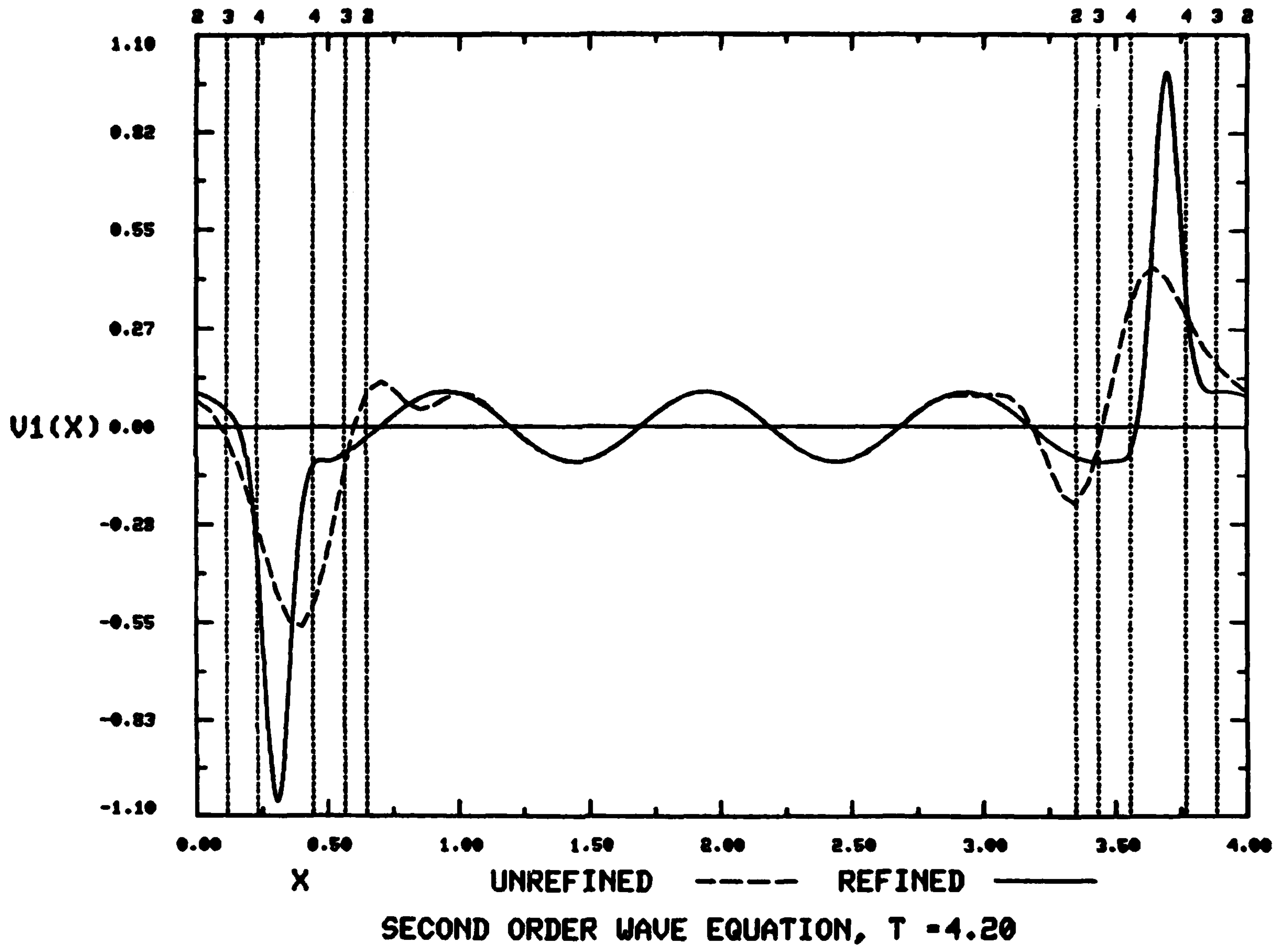
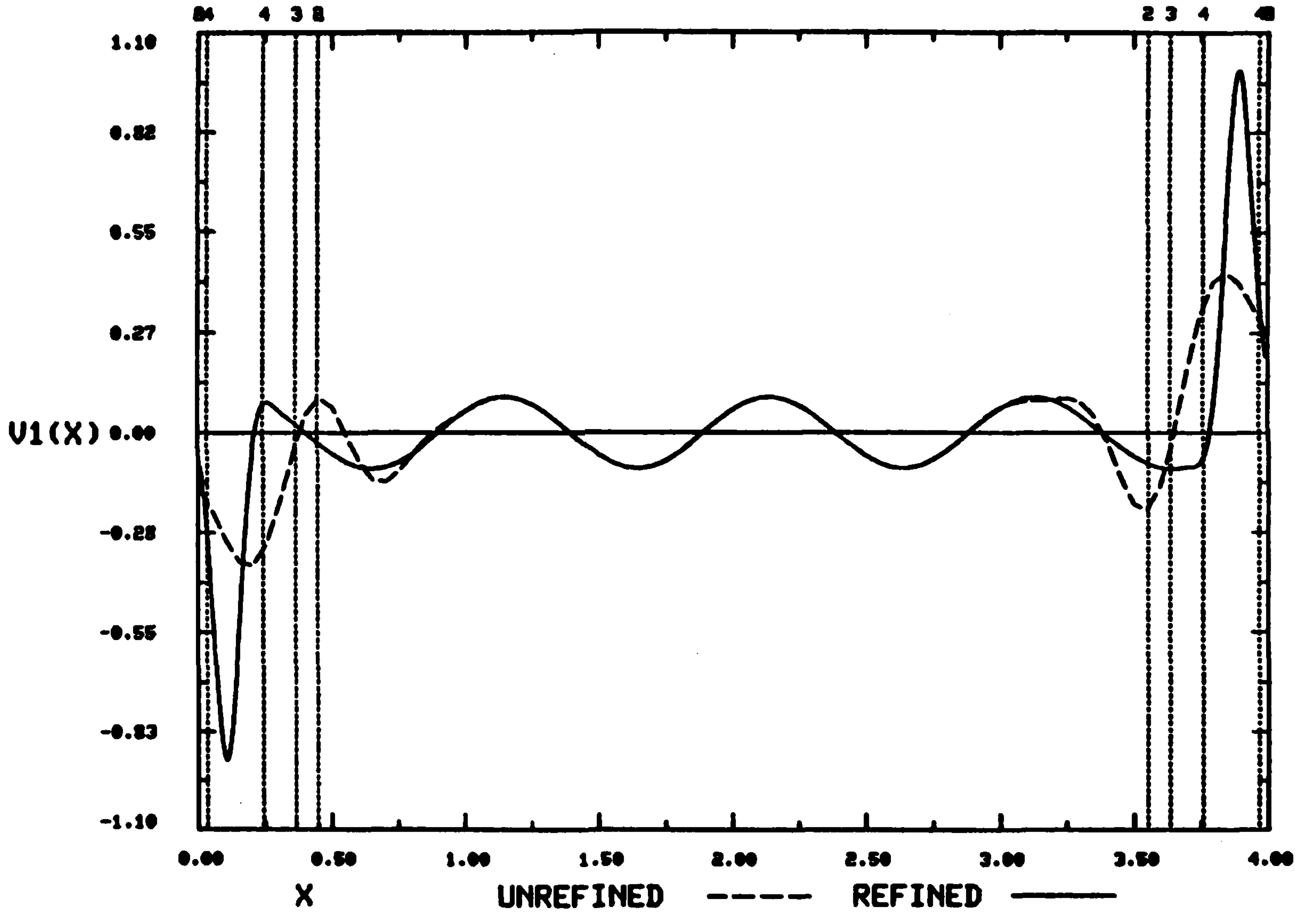
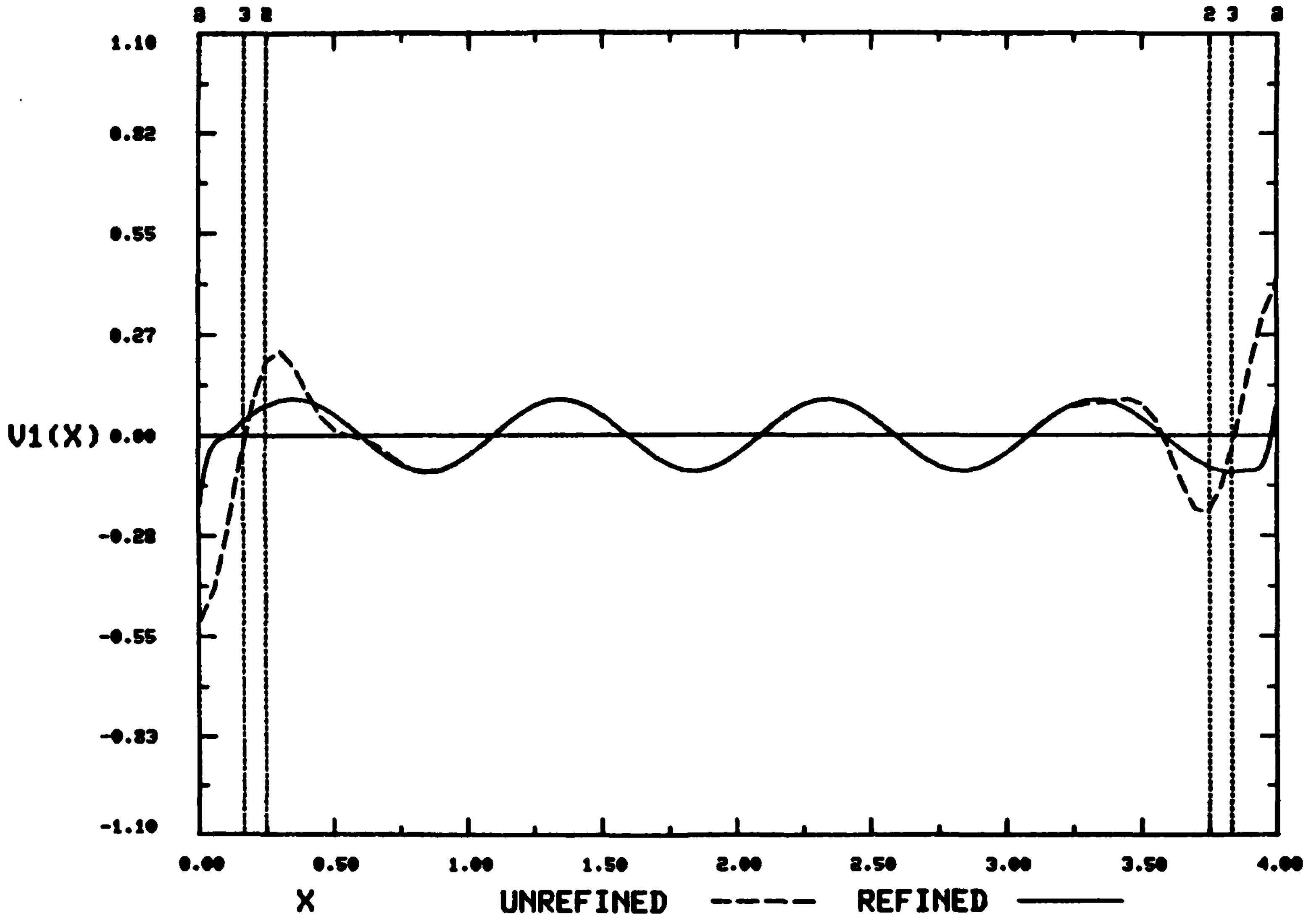

2 3 4 4 3 2
2 3 4 4 3 2
1.10
0.82
0.55
0.27
U1(X) 0.00
-0.28
-0.55
-0.83
-1.10
0.00
0.50
1.00
1.50
2.00
2.50
3.00
3.50
4.00
X
UNREFINED ---- REFINED ——
SECOND ORDER WAVE EQUATION, T =4.20

Figure 5(1)



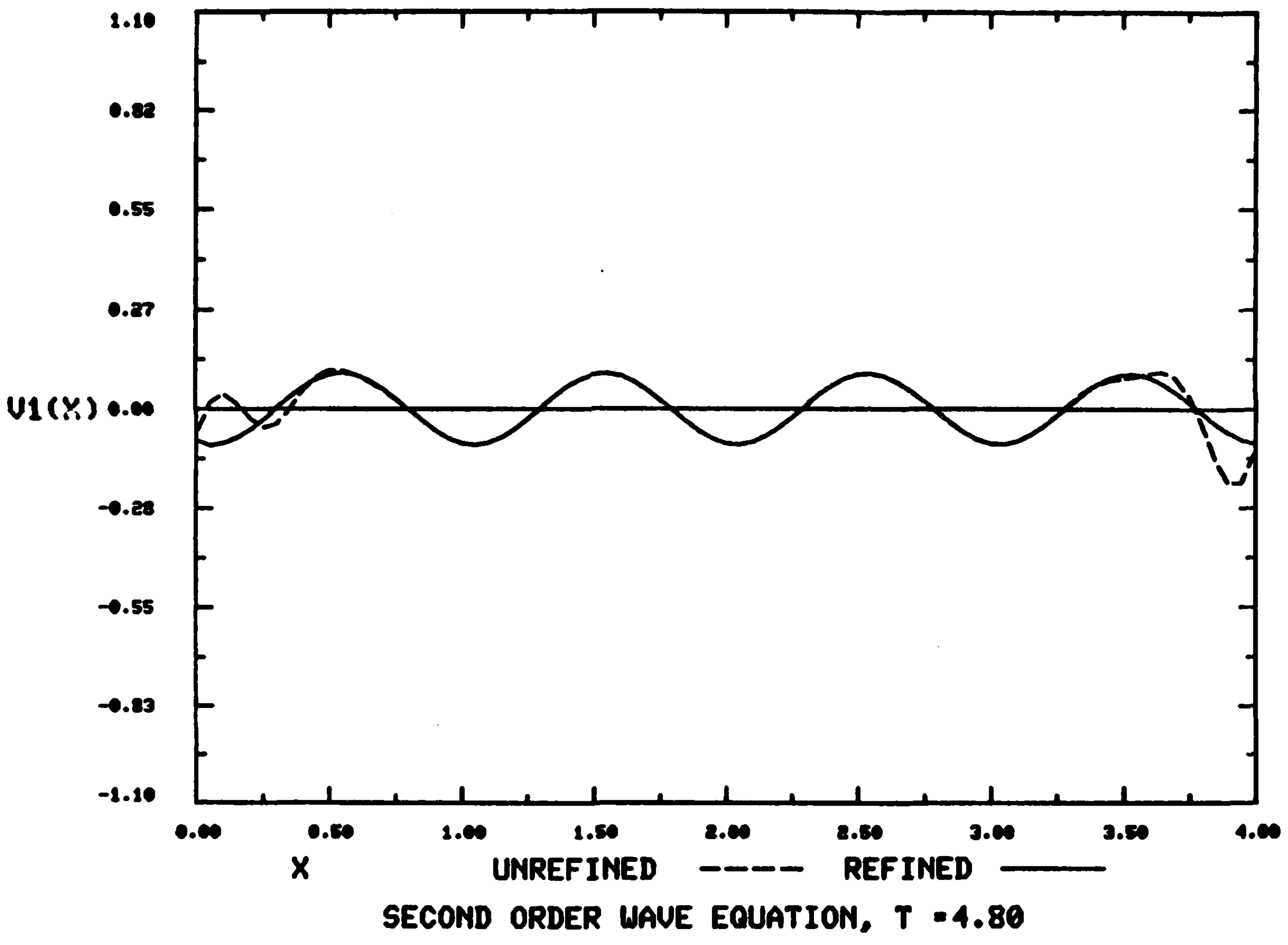
SECOND ORDER WAVE EQUATION, T =4.40

Figure 5(m)



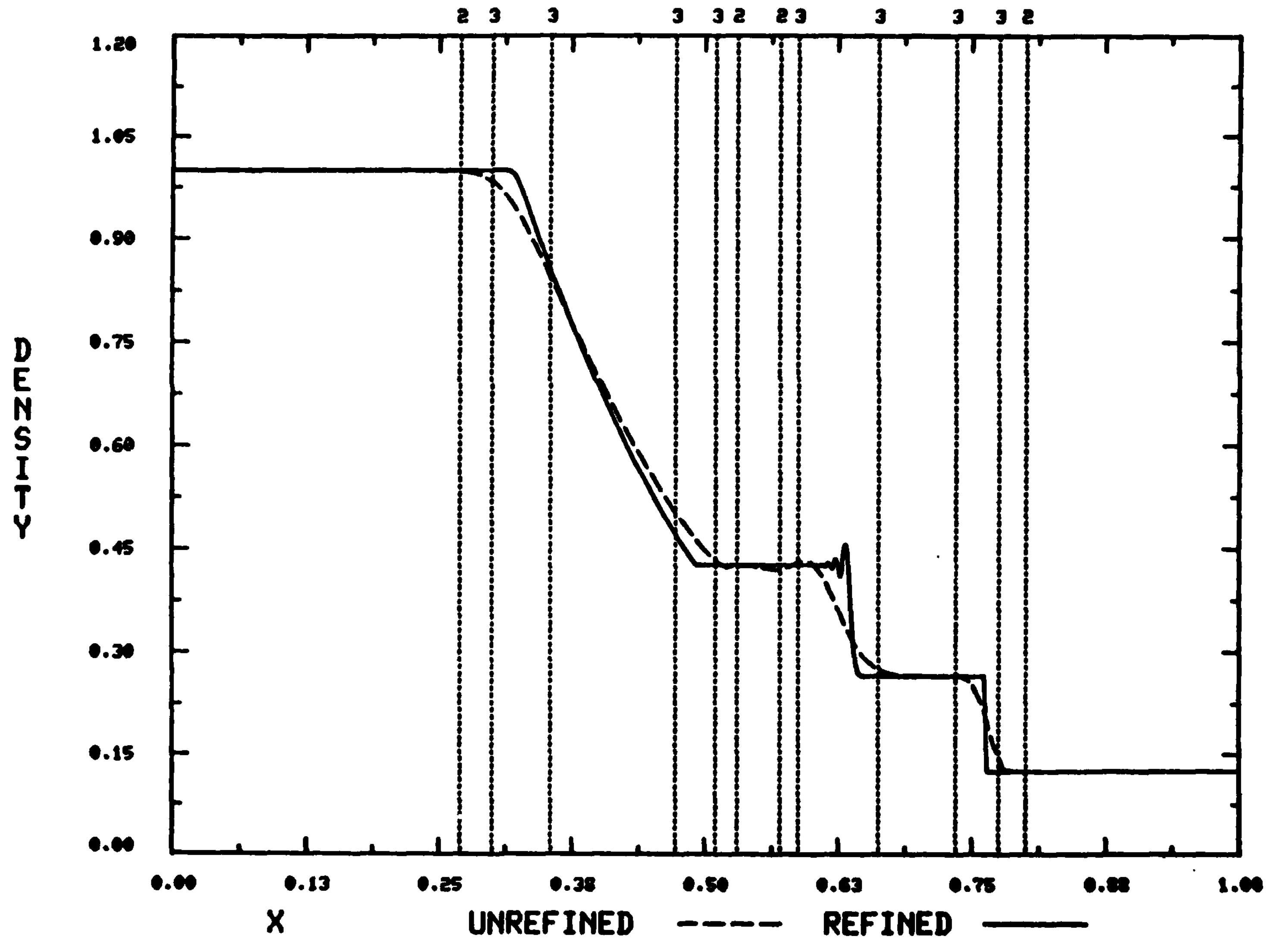
SECOND ORDER WAVE EQUATION, T =4.60

Figure 5(n)



SECOND ORDER WAVE EQUATION, T =4.80

Figure 5(o)



RIEMANN SHOCK TUBE PROBLEM, T =0.15

Figure 6(a)

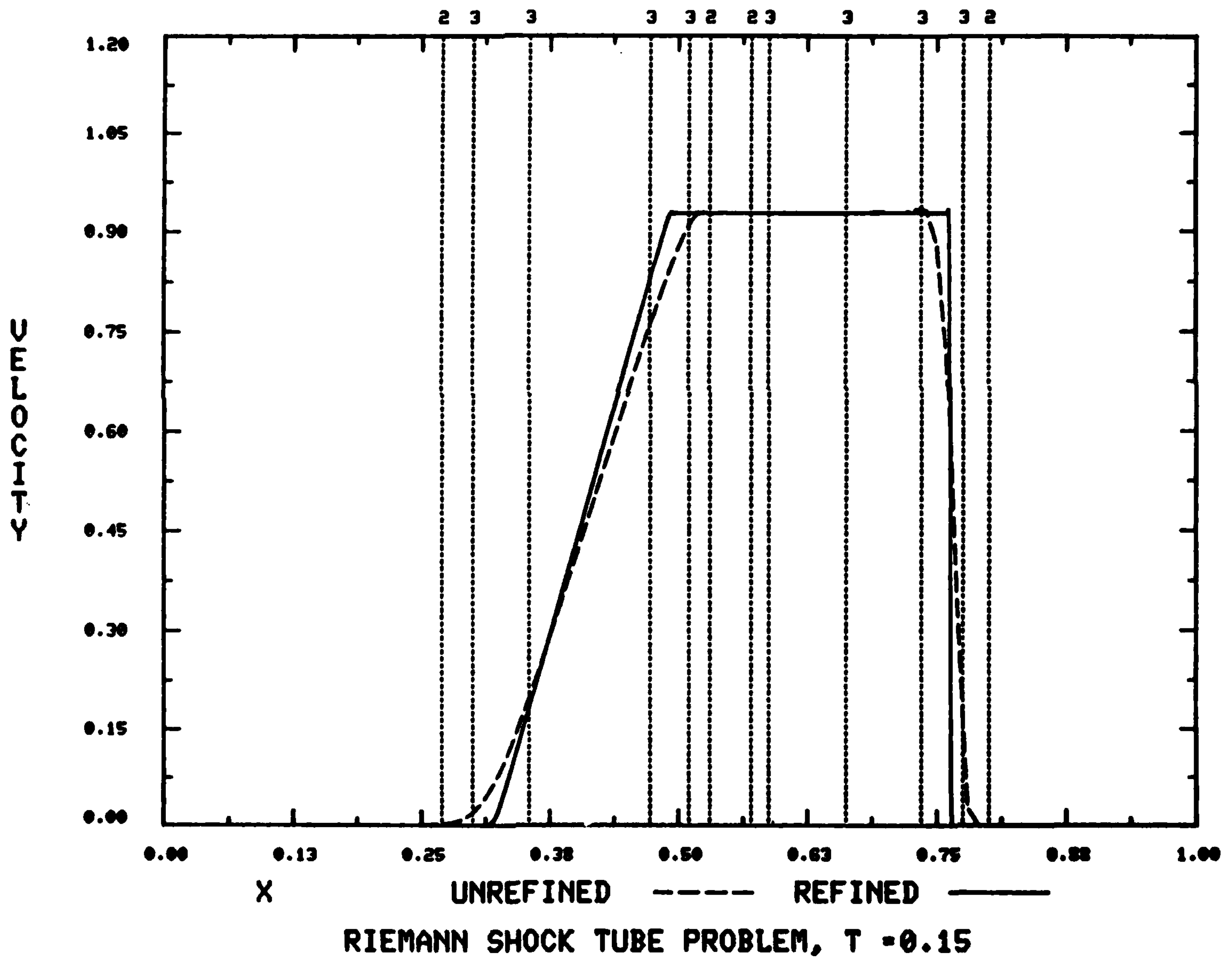
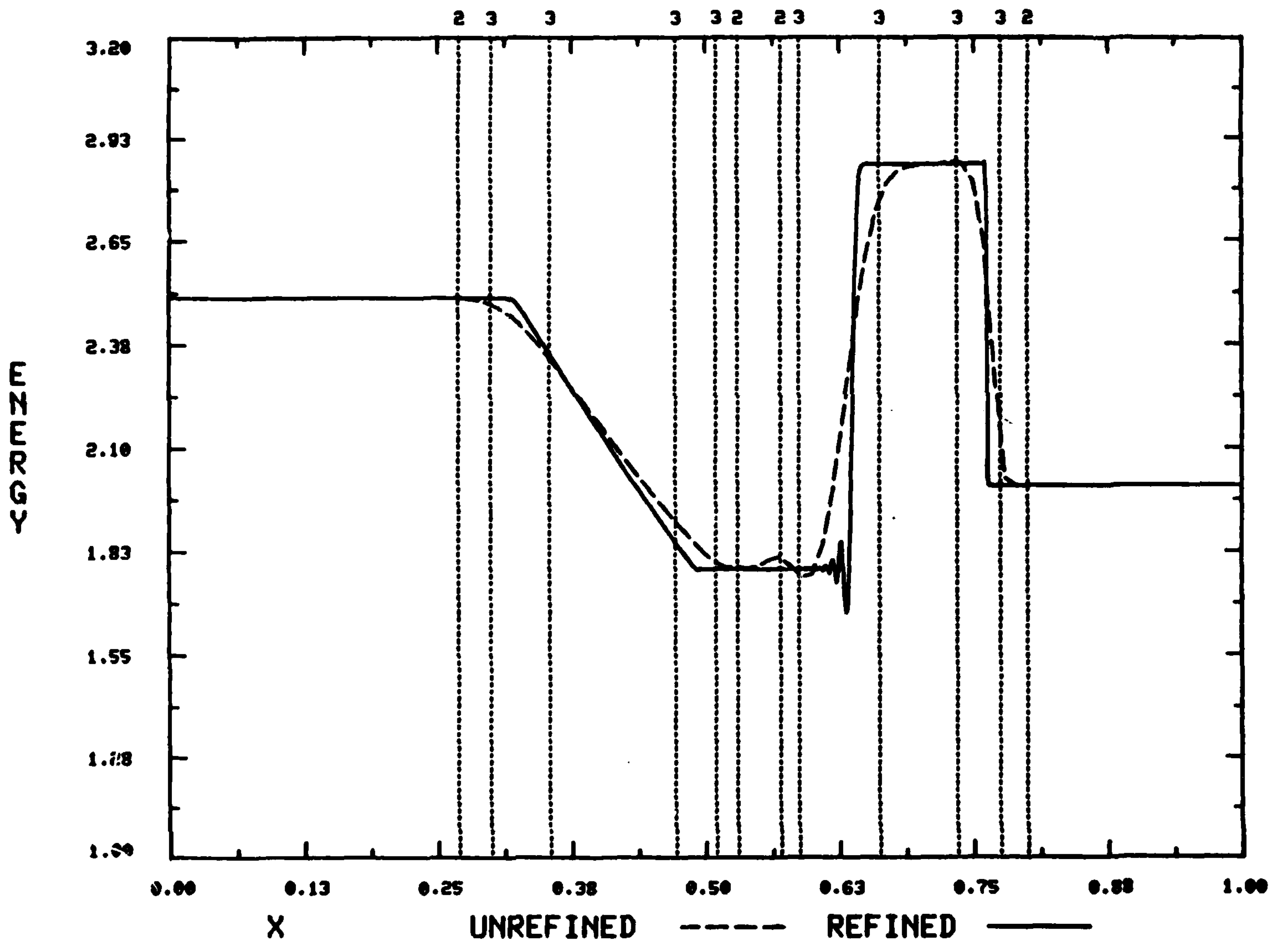

2 3 3 3 3 2 2 3 3 3 3 2
1.20
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00
VELOCITY
0.00
0.13
0.25
0.38
0.50
0.63
0.75
0.88
1.00
X
UNREFINED
REFINED
RIEMANN SHOCK TUBE PROBLEM, T =0.15

Figure 6(b)



RIEMANN SHOCK TUBE PROBLEM, T =0.15

Figure 6(d)

FIL